

Ecotone

Business first



**November 2023,
PHPCon Poland**

Agenda

Agenda



Resilient Messaging

Agenda



Resilient Messaging



Building Blocks

Agenda



Resilient Messaging

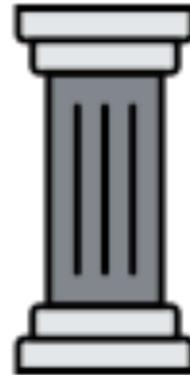


Building Blocks



Testing Messaging

**Prepare the exercise
wifi pass: verdevilla**



**[https://github.com/dgafka/
phpcon-2023-ecotone-
workshop](https://github.com/dgafka/phpcon-2023-ecotone-workshop)**

Let's discuss Ecotone, but first...



focus on the...

Business

focus on the...

Business

*As this is what
brings
the value*

Common Application challenges to solve:

Common Application challenges to solve:



Configuration & Bootstrap code

Common Application challenges to solve:

-  Configuration & Bootstrap code
-  Integration & Boilerplate code

Common Application challenges to solve:

-  Configuration & Bootstrap code
-  Integration & Boilerplate code
-  Resiliency & Recoverability code

Common Application challenges to solve:

- ✓ Configuration & Bootstrap code
- ✓ Integration & Boilerplate code
- ✓ Resiliency & Recoverability code

*This is not
Business related code*

Business Oriented Architecture



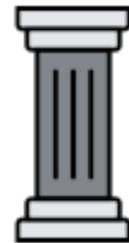
Resilient Messaging



Declarative Configuration



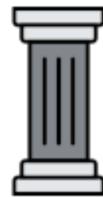
Resilient Messaging



Building Blocks



Declarative Configuration



Resilient Messaging

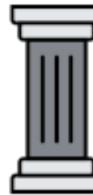


Business Oriented Code

Building Blocks



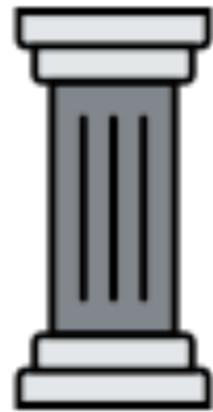
Declarative Configuration



Resilient Messaging



Messaging



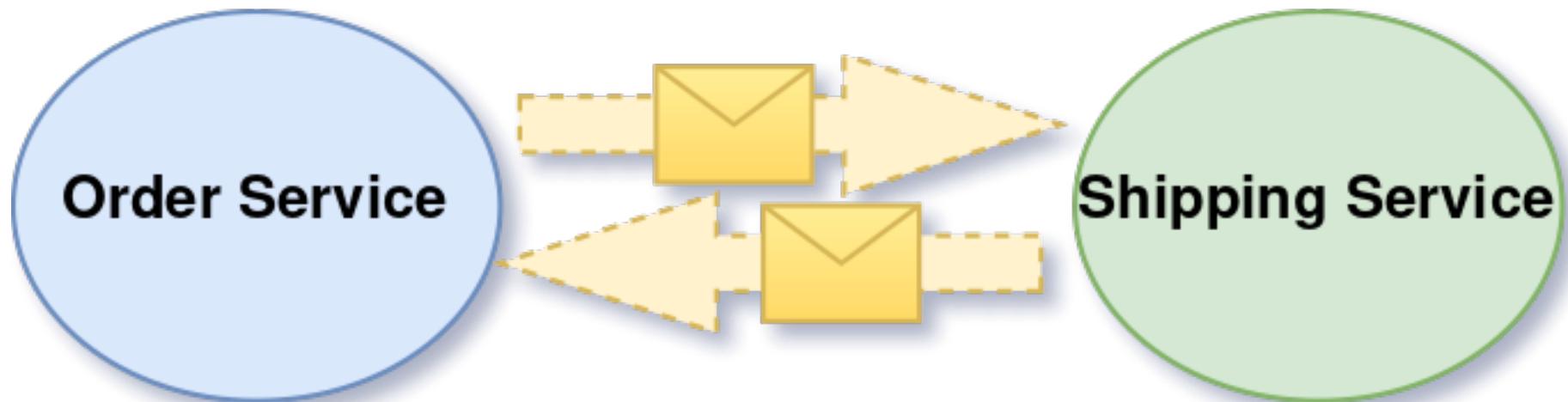


**“Object oriented programming,
gets people to focus on the lesser
idea.**

The big idea is Messaging.”

~ Alan Kay

Messaging is communication



Message is data record



```
interface Message
{
    public function getHeaders(): MessageHeaders;

    public function getPayload(): mixed;
}
```

Command Messages

```
final readonly class PlaceOrder
{
    public function __construct(
        public UuidInterface $orderId,
        public UuidInterface $productId
    ) {}

}
```

Event Messages

```
final readonly class OrderWasPlaced
{
    public function __construct(
        public UuidInterface $orderId
    ) {}

}
```

Sending Messages

```
final class OrderController
{
    public function __construct(private readonly CommandBus $commandBus) {}

    public function placeOrder(Request $request): Response
    {
        $command = $this->prepareCommandFromRequest($request);

        $this->commandBus->send($command);

        return new Response();
    }
}
```

Sending Messages

```
final class OrderController
{
    public function __construct(private readonly CommandBus $commandBus) {}

    public function placeOrder(Request $request): Response
    {
        $command = $this->prepareCommandFromRequest($request);

        $this->commandBus->send($command);

        return new Response();
    }
}
```

Sending Messages

```
final class OrderController
{
    public function __construct(private readonly CommandBus $commandBus) {}

    public function placeOrder(Request $request): Response
    {
        $command = $this->prepareCommandFromRequest($request);

        $this->commandBus->send($command);
        return new Response();
    }
}
```



Sending Messages

```
final class OrderController
{
    public function __construct(private readonly CommandBus $commandBus) {}

    public function placeOrder(Request $request): Response
    {
        $command = $this->prepareCommandFromRequest($request);

        $this->commandBus->send($command);

        return new Response();
    }
}
```

```
$this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
```

Sending Messages

```
final class OrderController
{
    public function __construct(private readonly CommandBus $commandBus) {}

    public function placeOrder(Request $request): Response
    {
        $command = $this->prepareCommandFromRequest($request);

        $this->commandBus->send($command);

        return new Response();
    }
}
```



```
$this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
```

Handling Messages

```
final class OrderService
{
    #[CommandHandler]
    public function placeOrder(PlaceOrder $command): void
    {
        $order = Order::create($command->orderId, $command->productId);

        $this->orderRepository->save($order);
    }
}
```

Handling Messages

```
final class OrderService
{
    #[CommandHandler]
    public function placeOrder(PlaceOrder $command): void
    {
        $order = Order::create($command->orderId, $command->productId);

        $this->orderRepository->save($order);
    }
}
```

Handling Messages

```
final class OrderService
{
    #[CommandHandler]
    public function placeOrder(PlaceOrder $command): void
    {
        $order = Order::create($command->orderId, $command->productId);

        $this->orderRepository->save($order);
    }
}
```

Handling Messages

```
final class OrderService
{
    #[CommandHandler]
    public function placeOrder(PlaceOrder $command): void
    {
        $order = Order::create($command->orderId, $command->productId);

        $this->orderRepository->save($order);
    }

    #[EventHandler] ←
    public function whenOrderWasPlaced(
        OrderWasPlaced $event, ShippingService $shippingService
    ): void
    {
        $order = $this->orderRepository->get($event->orderId);

        $shippingService->ship($order);
    }
}
```

Handling Messages

```
final class OrderService
{
    #[CommandHandler]
    public function placeOrder(PlaceOrder $command): void
    {
        $order = Order::create($command->orderId, $command->productId);

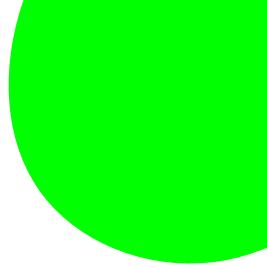
        $this->orderRepository->save($order);
    }

#[EventHandler]
public function whenOrderWasPlaced(
    OrderWasPlaced $event, ShippingService $shippingService
): void
{
    $order = $this->orderRepository->get($event->orderId);

    $shippingService->ship($order);
}
```



Placing order flow





Placing order flow

- Storing Order in database



Placing order flow

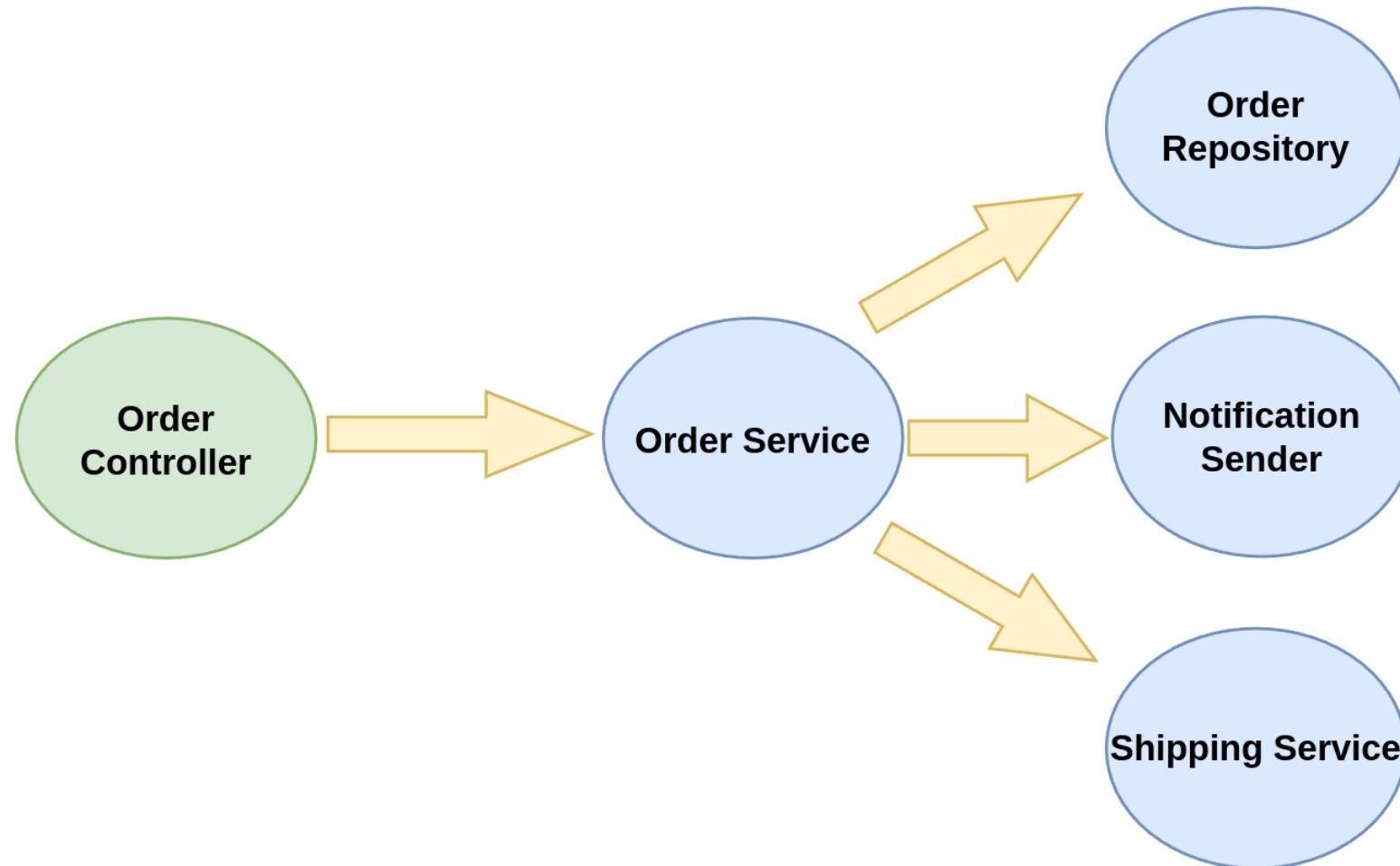
- Storing Order in database
- Sending confirmation e-mail to the customer with the order's summary



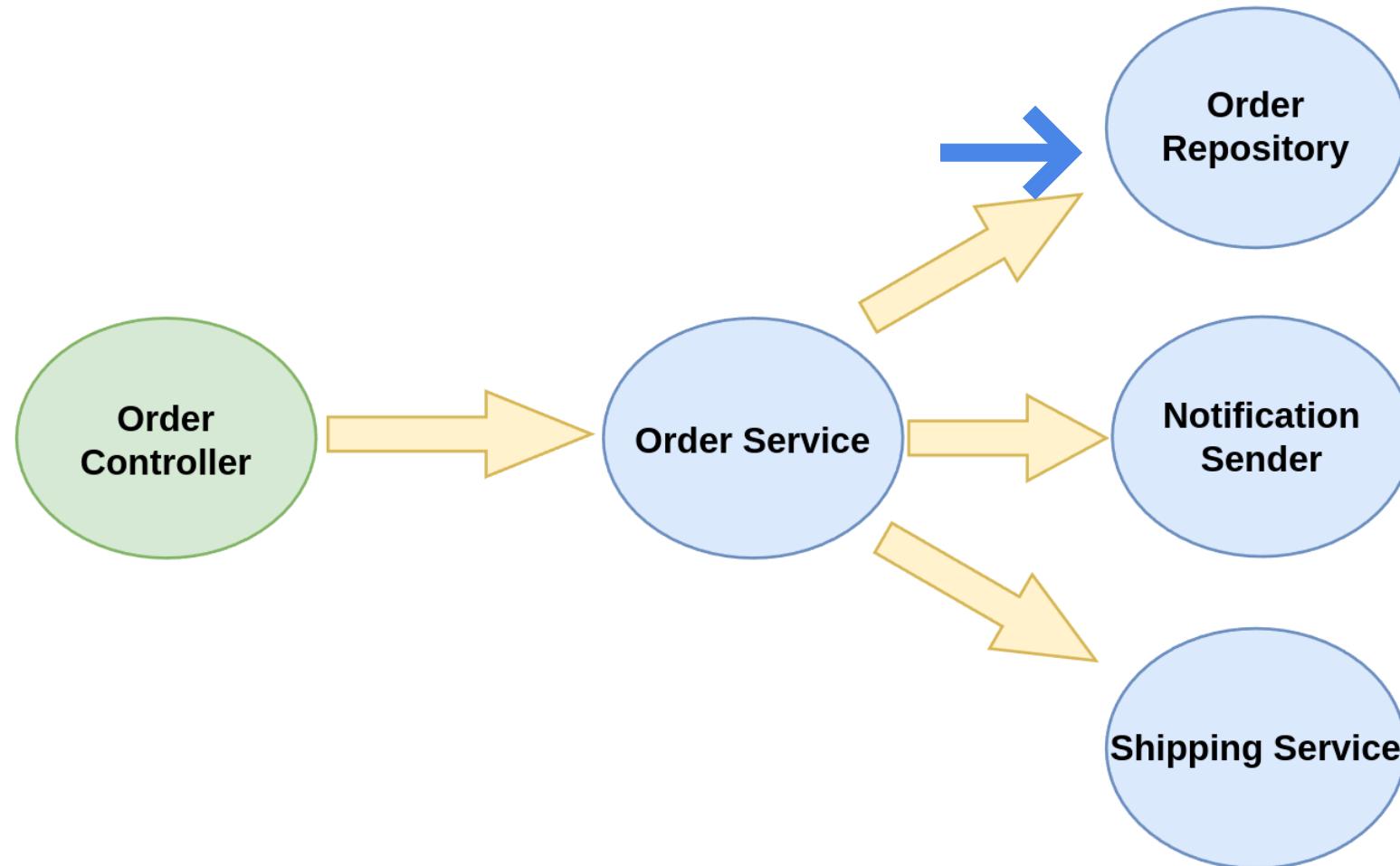
Placing order flow

- Storing Order in database
- Sending confirmation e-mail to the customer with the order's summary
- Starting delivery process by calling a Shipping Service over HTTP API

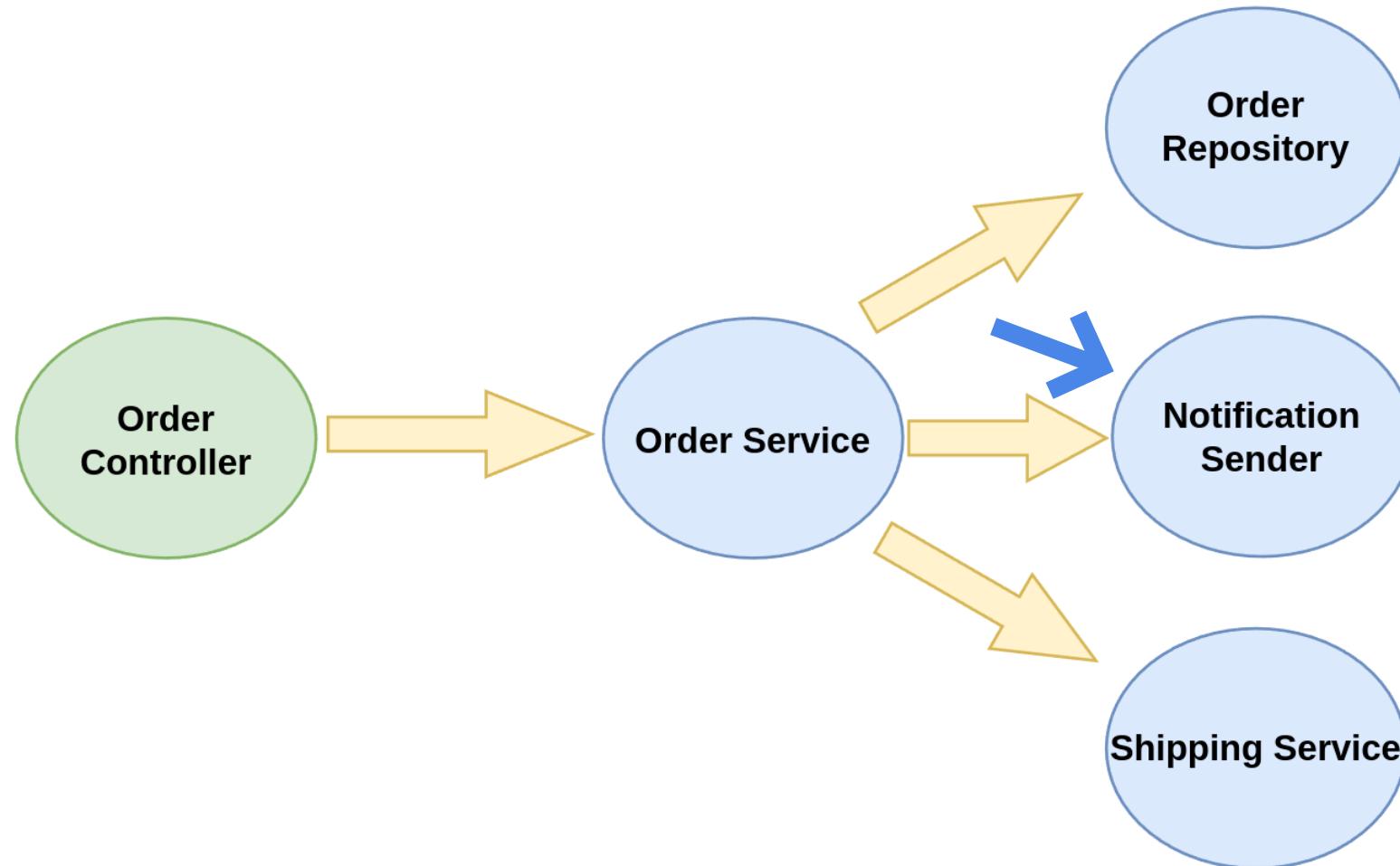
Non Message-Driven Architecture



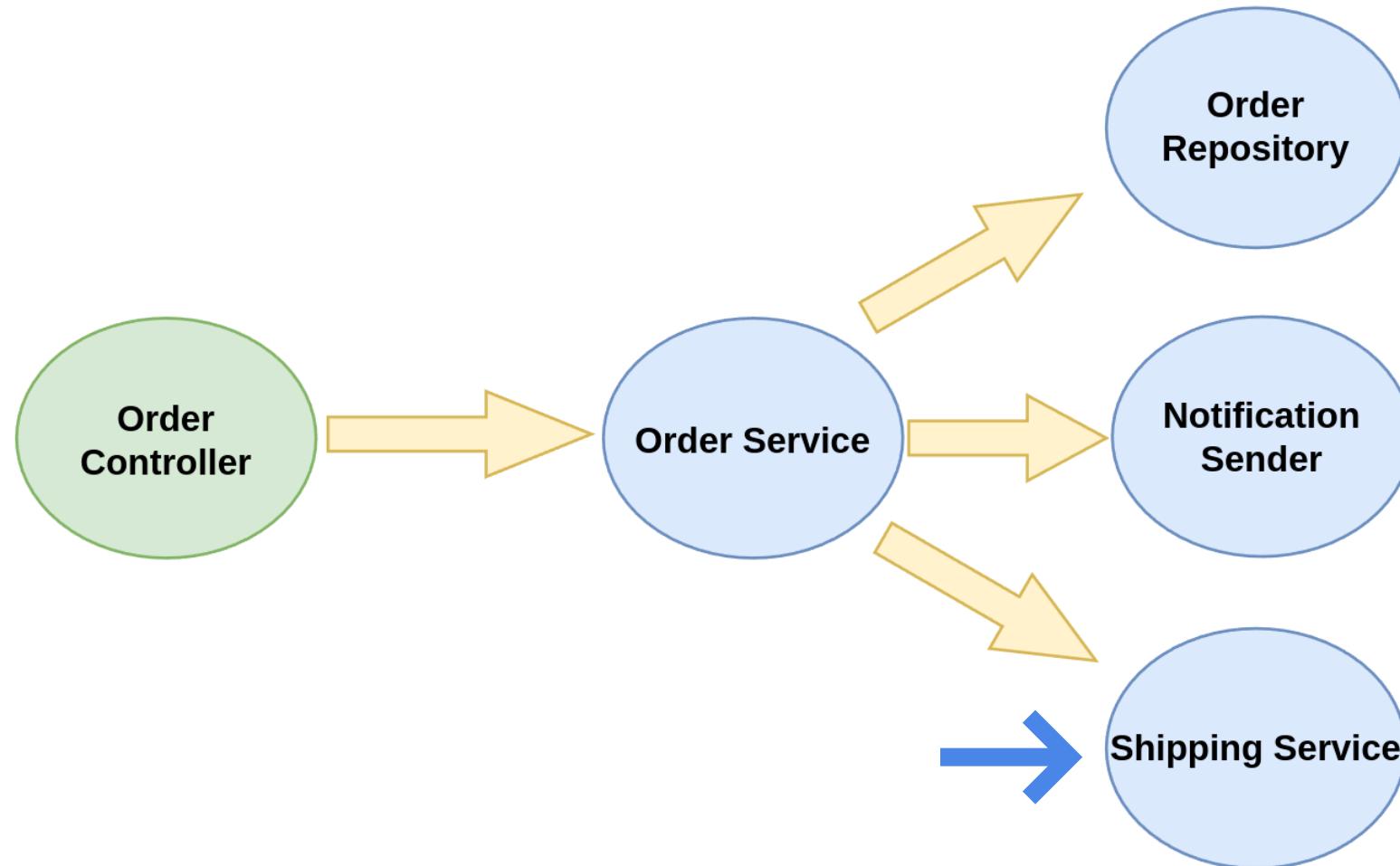
Non Message-Driven Architecture



Non Message-Driven Architecture



Non Message-Driven Architecture



Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order);

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order);
}
```

Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order);

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order);
}
```



Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order); 
}

/** Calling Shipping Service over HTTP, to deliver products */
$this->shippingService->shipOrderFor($order);
}
```

Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order);

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order);
}
```



Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order); ✓

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order);

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order);
}
```

Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order); ✓

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order); ✗

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order);
}
```

Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order); ✓

    /** Sending order confirmation notification */
    $this->notifcationSender->send($order); ✓

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order);
}
```

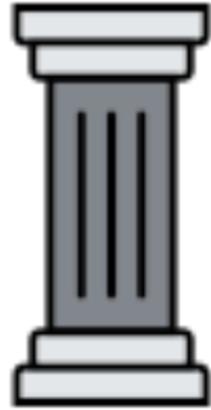
Non-Message Driven Architecture

```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order); ✓

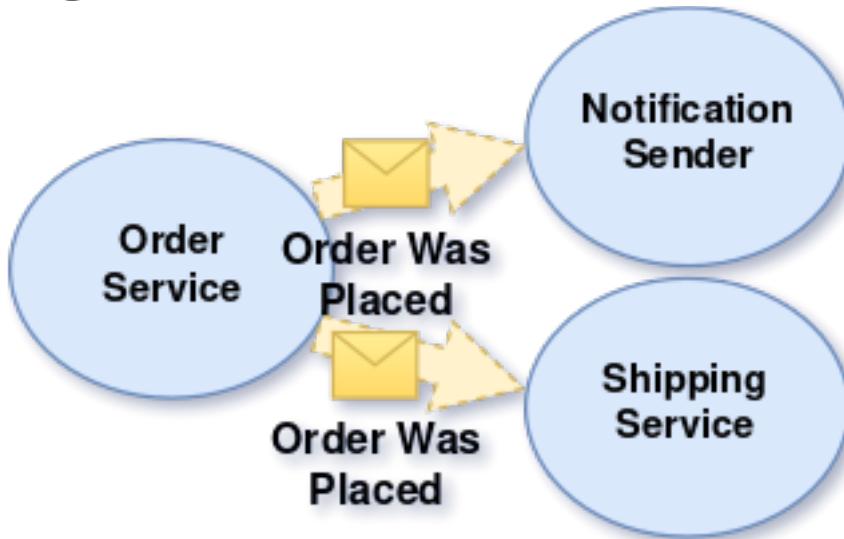
    /** Sending order confirmation notification */
    $this->notifcationSender->send($order); ✓

    /** Calling Shipping Service over HTTP, to deliver products */
    $this->shippingService->shipOrderFor($order); ✗
}
```

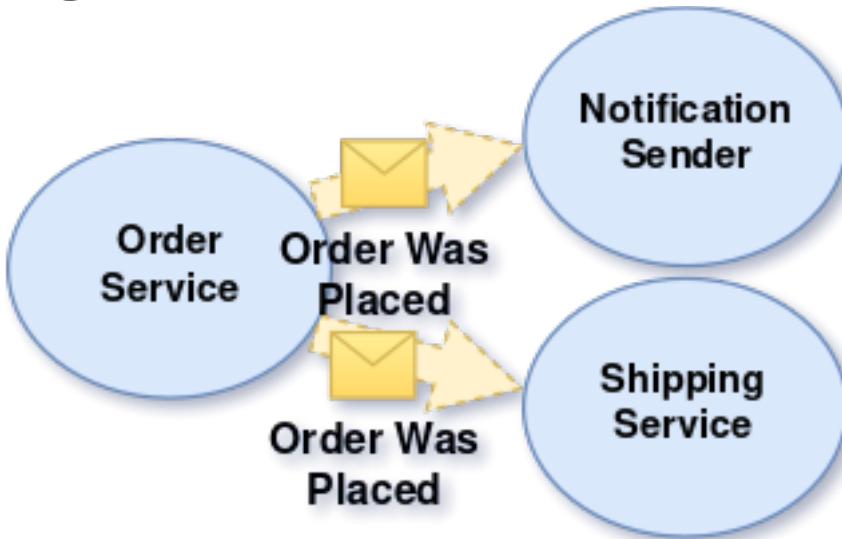
Resilient Messaging



Message-Driven Architecture

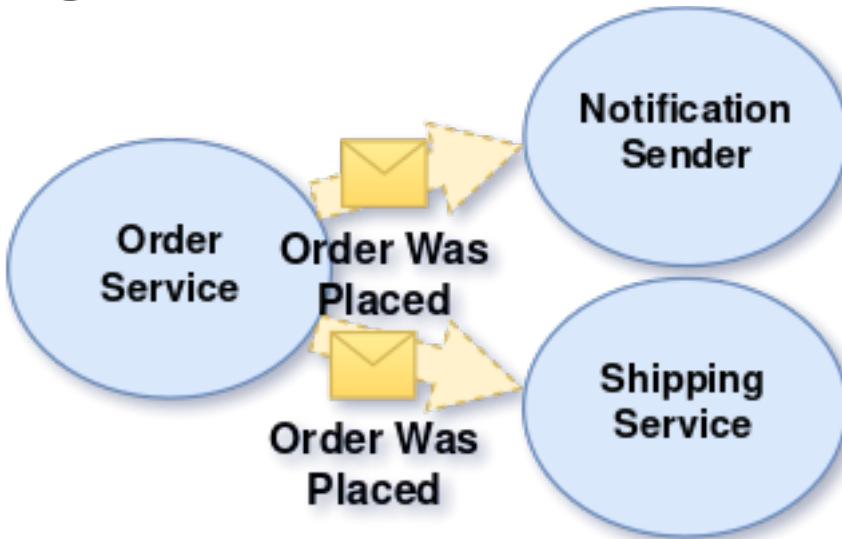


Message-Driven Architecture



```
public function placeOrder(  
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId  
): void  
{  
    $order = Order::create(  
        $userId, $shippingAddress,  
        $this->getProductDetails($productId), $this->clock  
    );  
    /** Storing order in database */  
    $this->orderRepository->save($order);  
  
    /** Publish event indicating that Order Was Placed */  
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));  
}
```

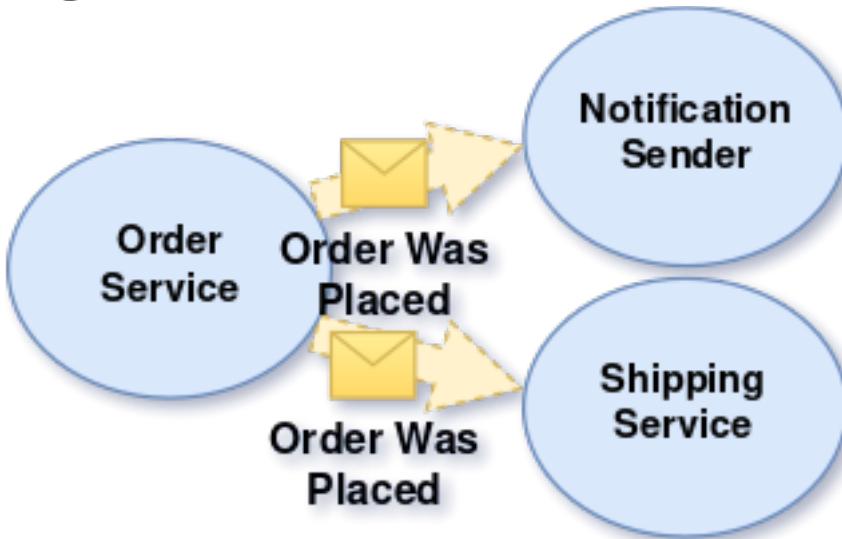
Message-Driven Architecture



```
public function placeOrder(
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId
): void
{
    $order = Order::create(
        $userId, $shippingAddress,
        $this->getProductDetails($productId), $this->clock
    );
    /* Storing order in database */
    $this->orderRepository->save($order); ←

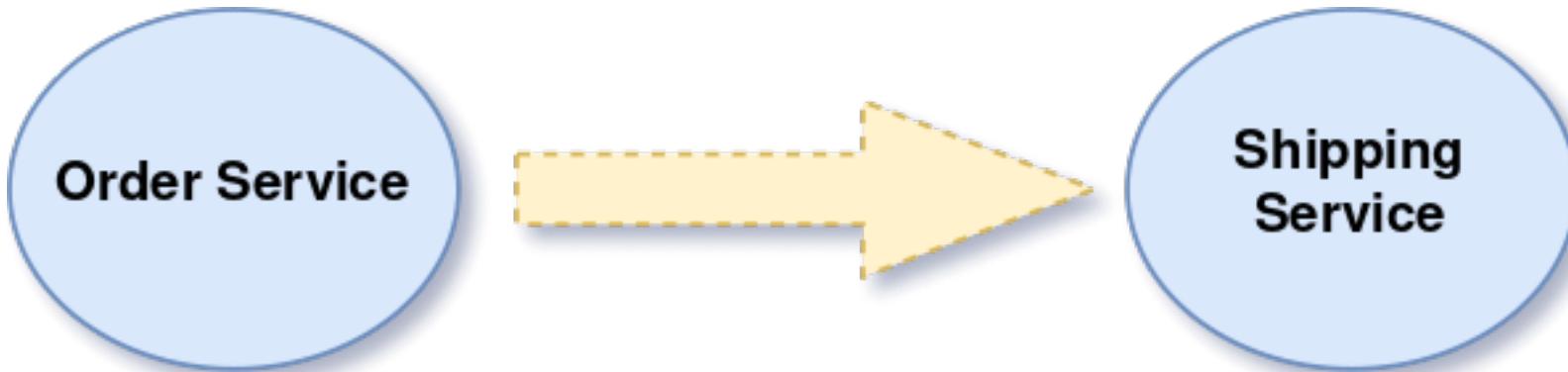
    /* Publish event indicating that Order Was Placed */
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
}
```

Message-Driven Architecture



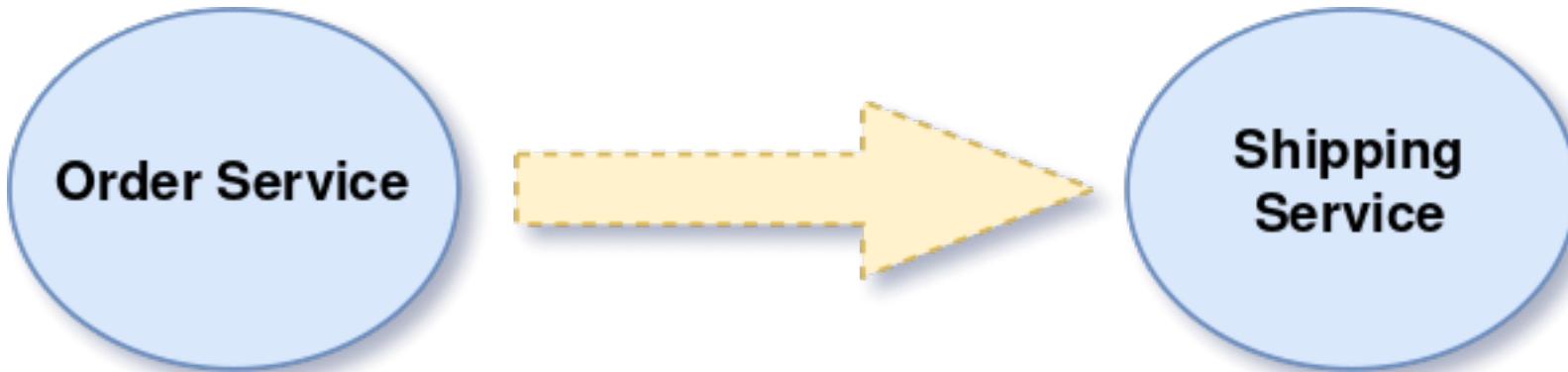
```
public function placeOrder(  
    UuidInterface $userId, ShippingAddress $shippingAddress, UuidInterface $productId  
): void  
{  
    $order = Order::create(  
        $userId, $shippingAddress,  
        $this->getProductDetails($productId), $this->clock  
    );  
    /** Storing order in database */  
    $this->orderRepository->save($order);  
  
    /** Publish event indicating that Order Was Placed */  
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));  
}
```

Subscribing to Order Was Placed



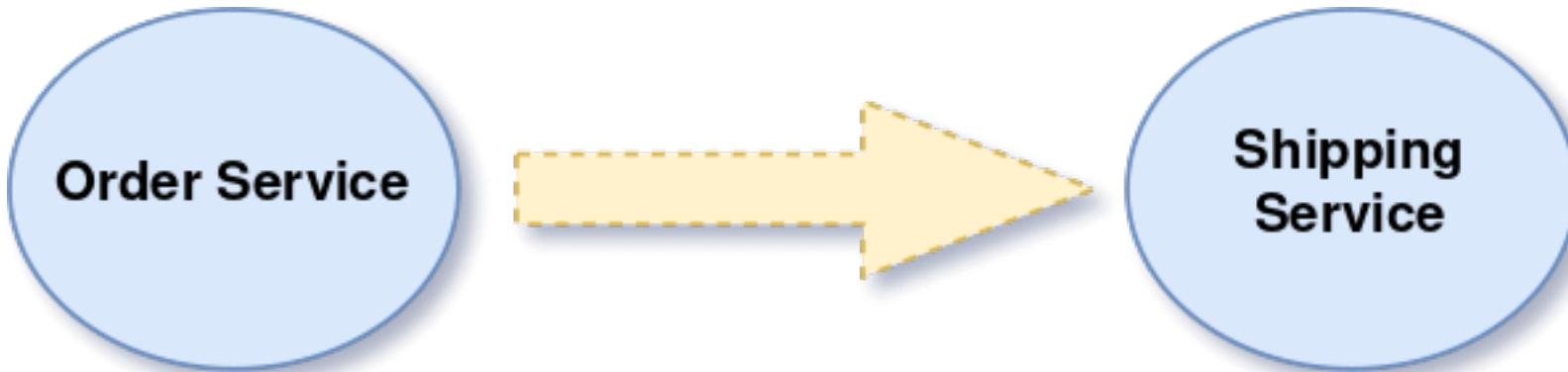
```
#[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Subscribing to Order Was Placed



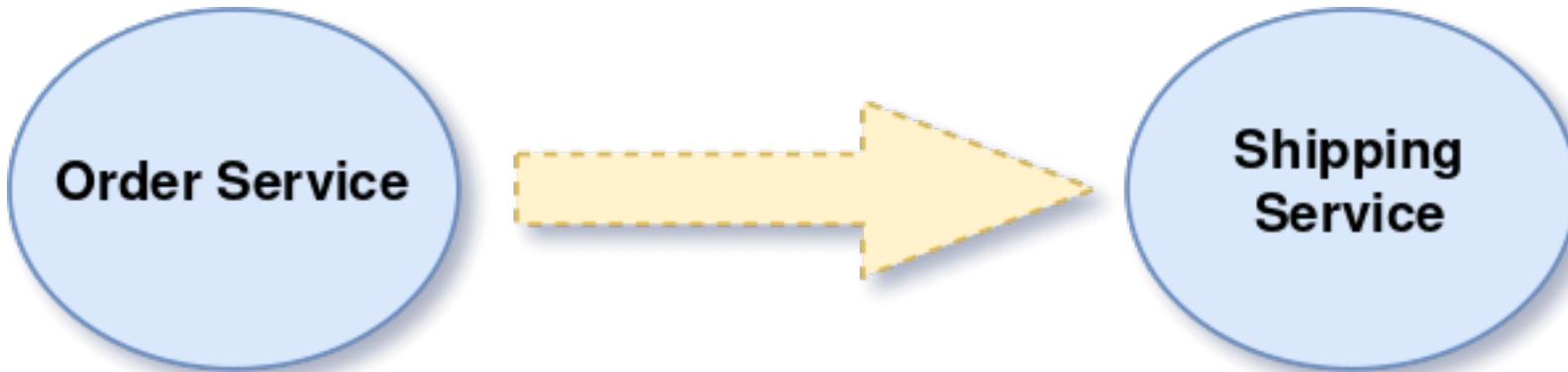
```
##[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Subscribing to Order Was Placed



```
##[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

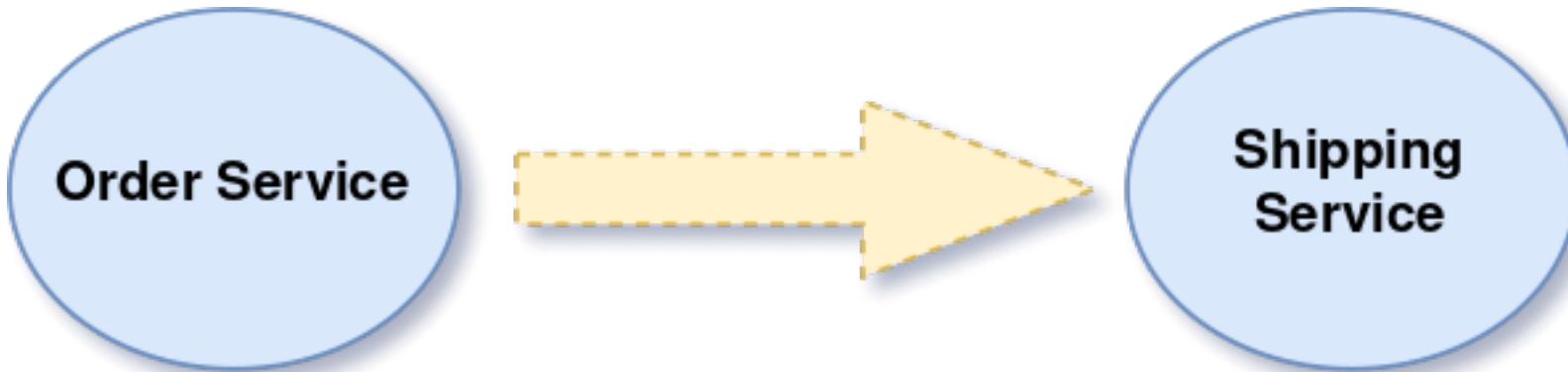
Subscribing to Order Was Placed



```
##[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

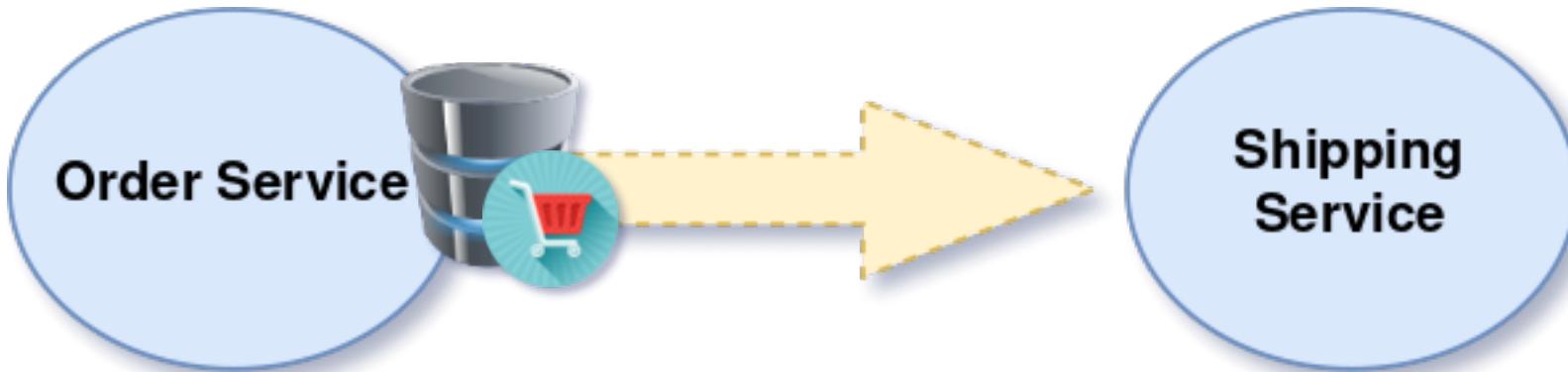
A blue arrow points from the word "Shipping" in the "Shipping Service" circle to the method call `$this->shippingService->shipOrderFor()` in the code snippet, indicating the target of the subscription.

Subscribing to Order Was Placed



```
#[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Subscribing to Order Was Placed



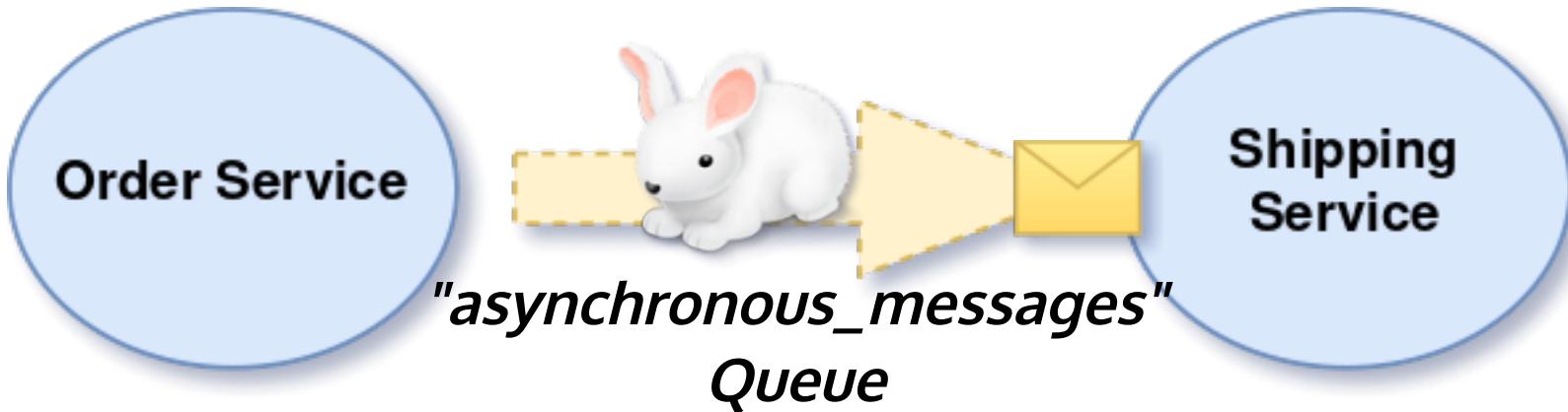
```
#[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Subscribing to Order Was Placed



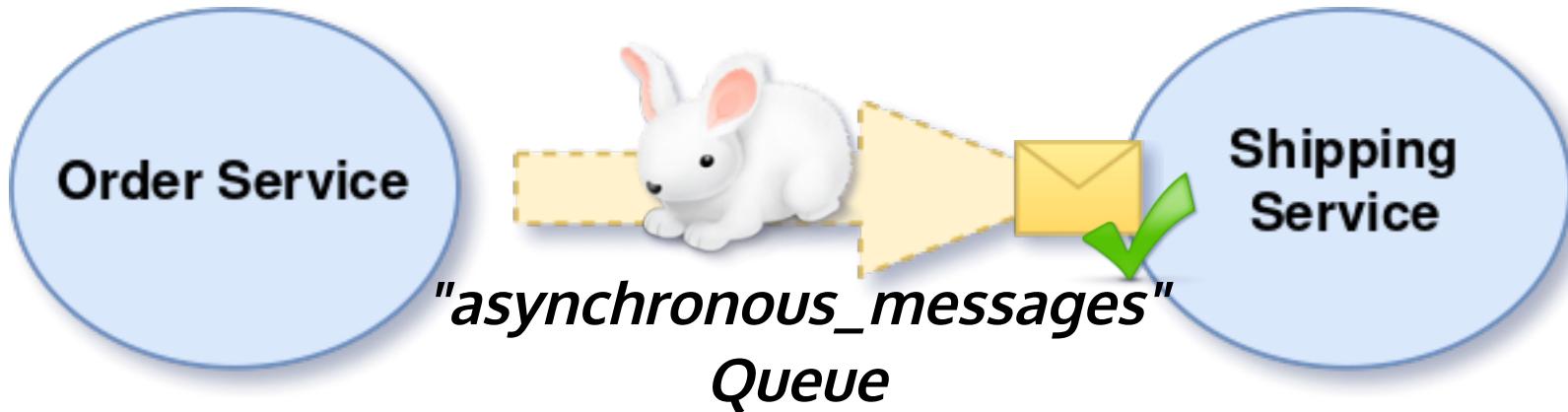
```
#[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Subscribing to Order Was Placed



```
#[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Subscribing to Order Was Placed



```
#[Asynchronous("asynchronous_messages")]
#[EventHandler(endpointId: "shipWhenOrderWasPlaced")]
public function when(OrderWasPlaced $event): void
{
    $this->shippingService->shipOrderFor(
        $this->orderRepository->getBy($event->orderId)
    );
}
```

Setting up asynchronous channel

```
final class MessageChannelConfiguration
{
    #[ServiceContext]
    public function asynchronousChannel()
    {
        return AmqpBackedMessageChannelBuilder::create(
            channelName: "asynchronous_messages"
        );
    }
}
```


Setting up asynchronous channel

```
final class MessageChannelConfiguration
{
    #[ServiceContext]
    public function asynchronousChannel()
    {
        return AmqpBackedMessageChannelBuilder::create(
            channelName: "asynchronous_messages"
        );
    }
}
```



Setting up asynchronous channel

```
final class MessageChannelConfiguration
{
    #[ServiceContext]
    public function asynchronousChannel()
    {
        return AmqpBackedMessageChannelBuilder::create(
            channelName: "asynchronous_messages"
        );
    }
}
```



Setting up asynchronous channel

```
final class MessageChannelConfiguration
{
    #[ServiceContext]
    public function asynchronousChannel()
    {
        return AmqpBackedMessageChannelBuilder::create(
            channelName: "asynchronous_messages"
        );
    }
}
```


Setting up asynchronous channel



RabbitMQ Channel

Setting up asynchronous channel



RabbitMQ Channel



Amazon SQS Channel

Setting up asynchronous channel



RabbitMQ Channel



Amazon SQS Channel



Redis Channel

Setting up asynchronous channel



RabbitMQ Channel



Amazon SQS Channel



Redis Channel



Database Channel

Setting up asynchronous channel



RabbitMQ Channel



Amazon SQS Channel



Redis Channel

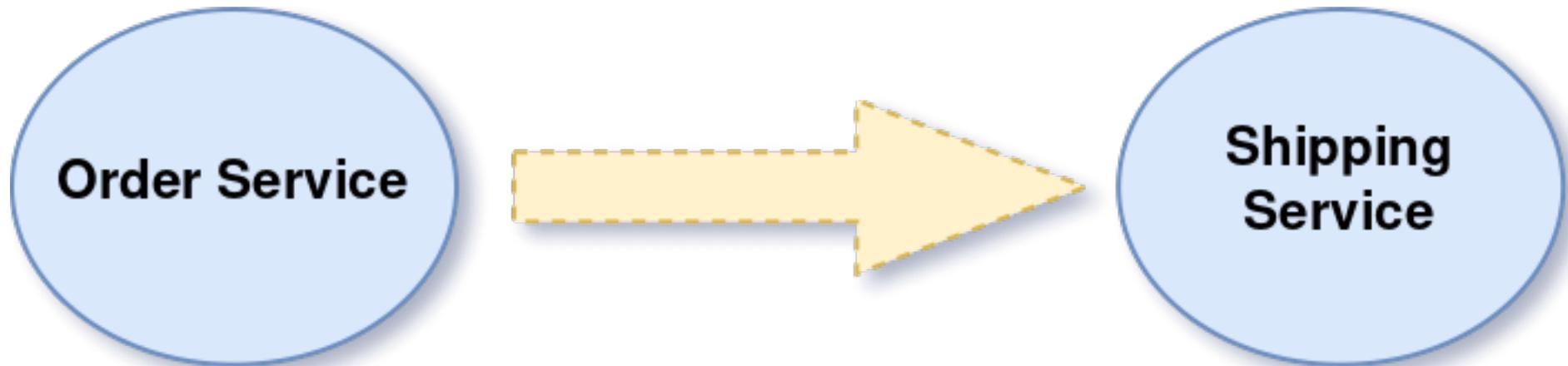


Database Channel

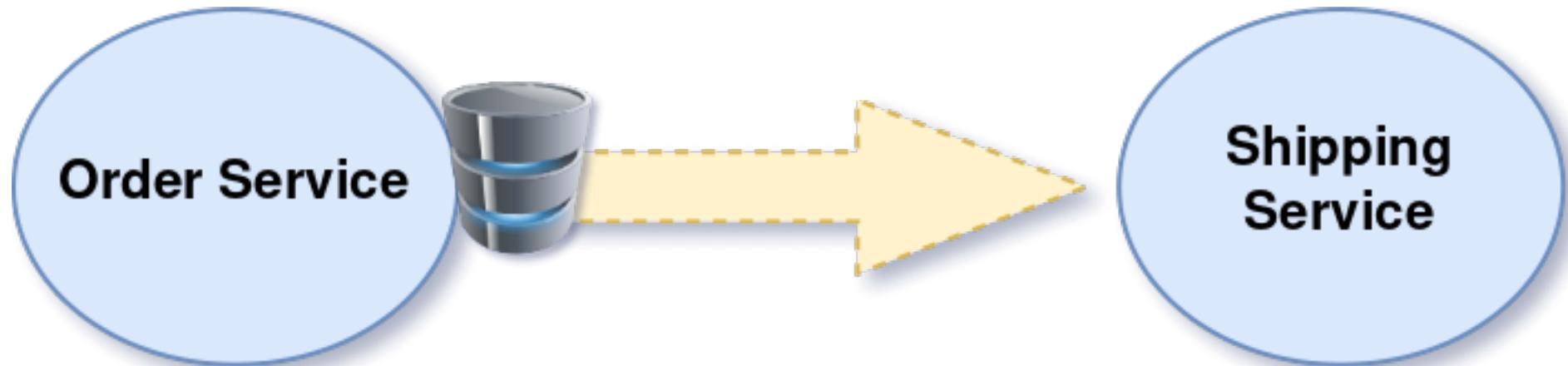


Symfony Messenger
Transport Channel

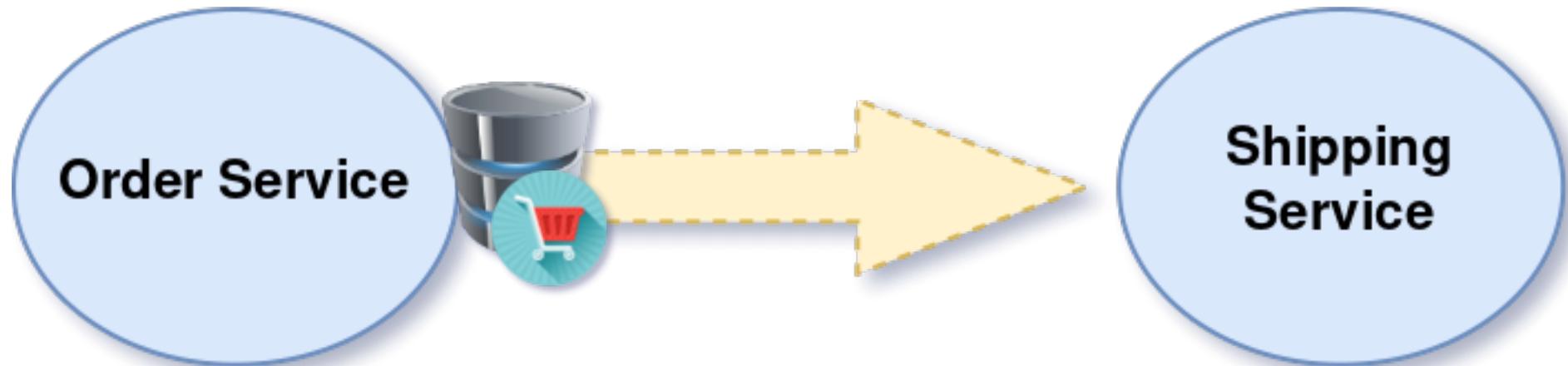
Data consistency



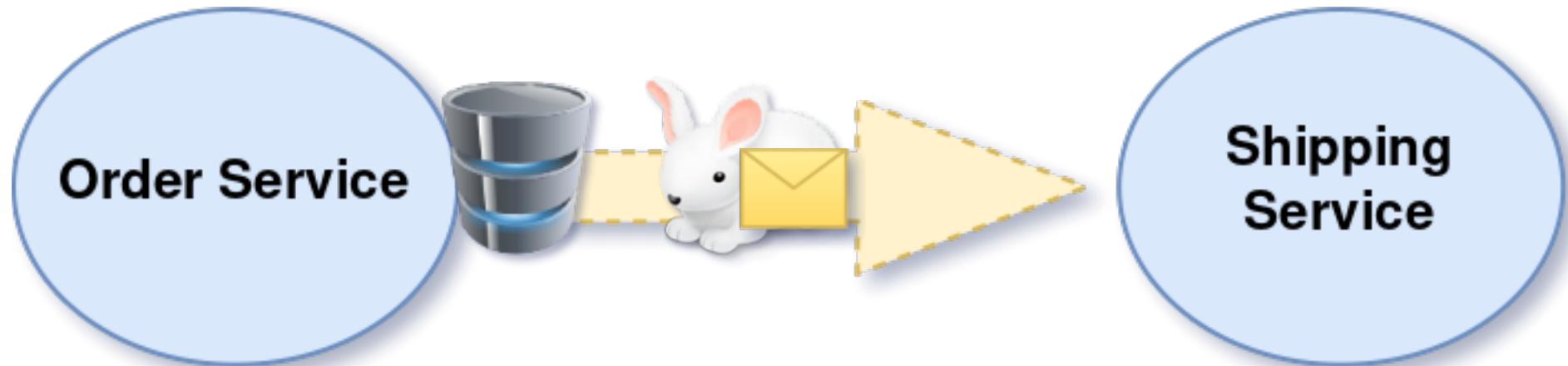
Data consistency



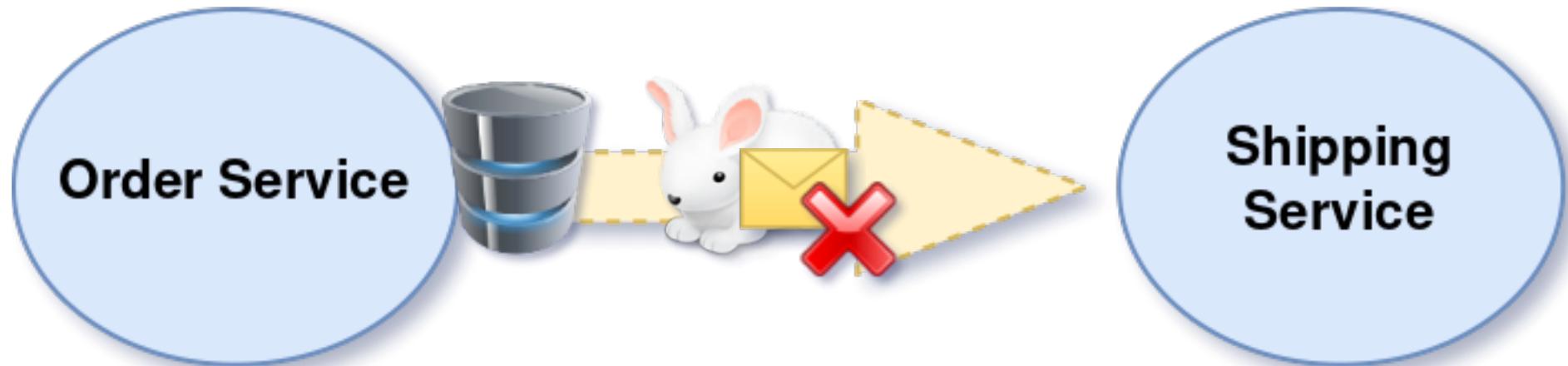
Data consistency



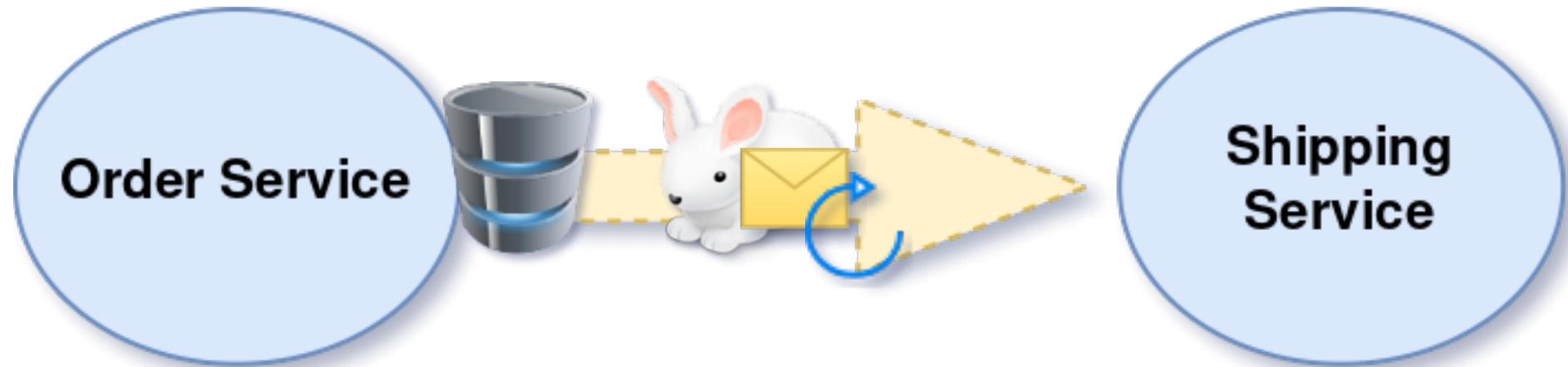
Data consistency



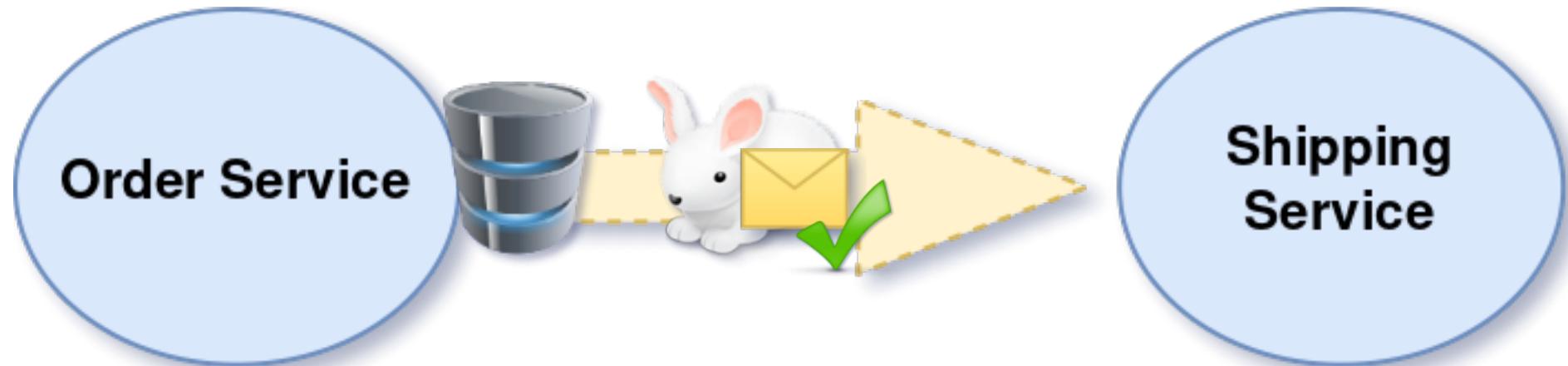
Data consistency



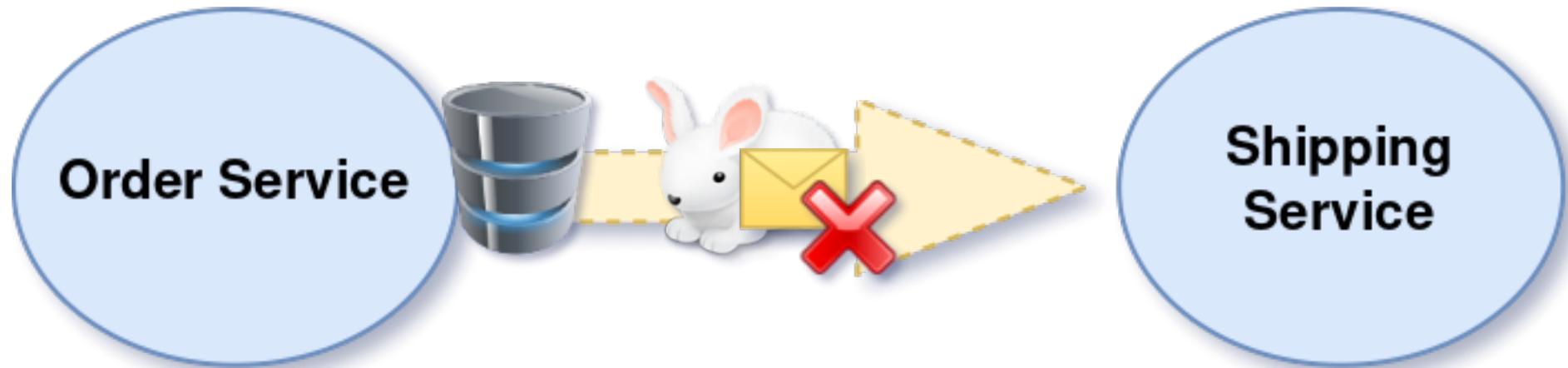
Data consistency



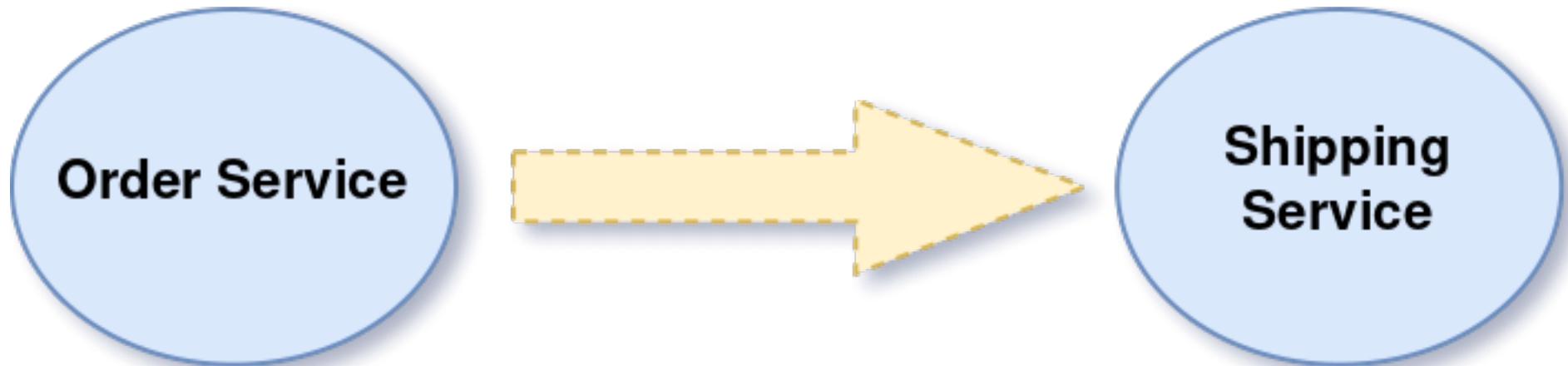
Data consistency



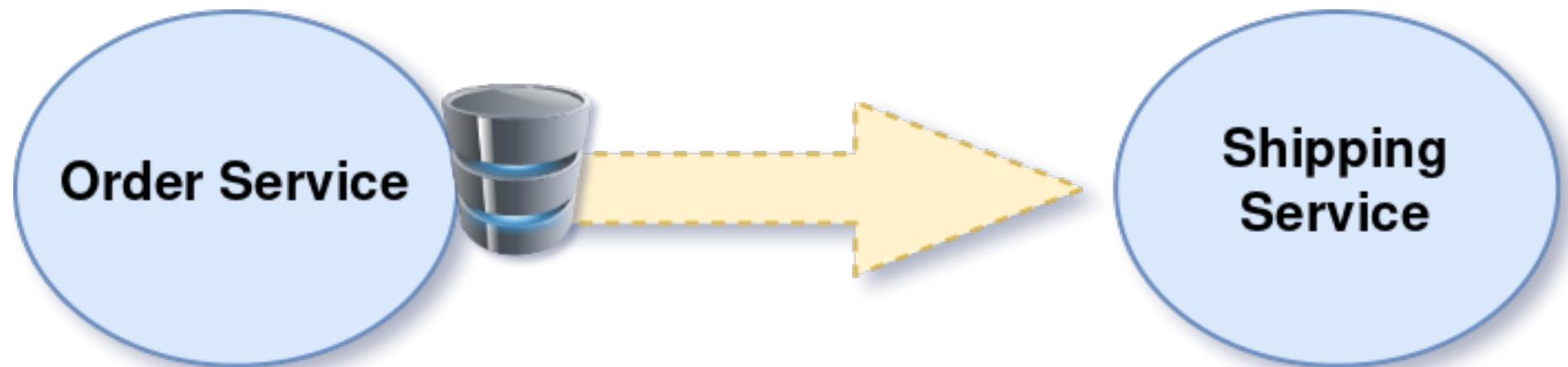
Data consistency



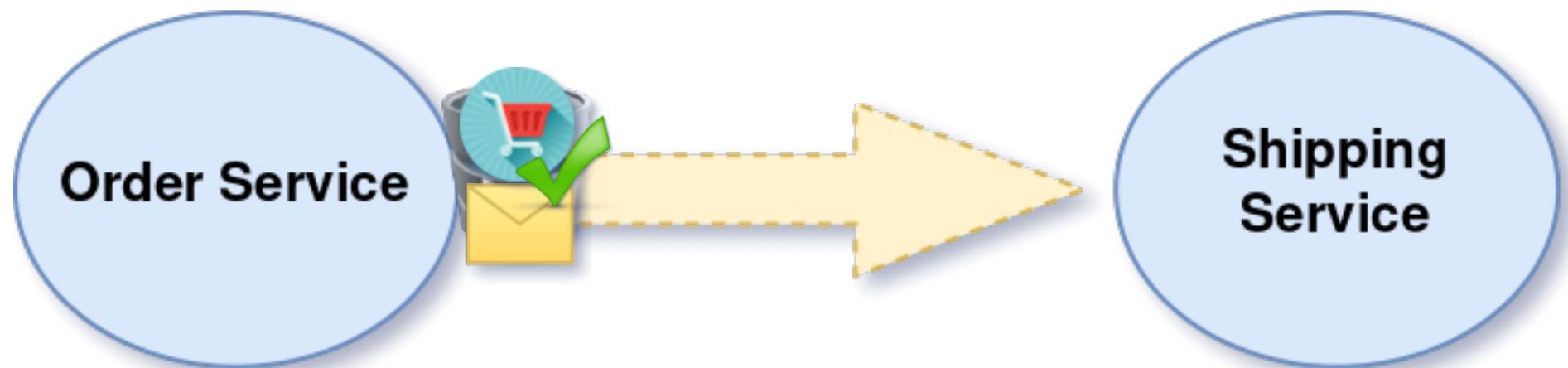
Outbox Pattern



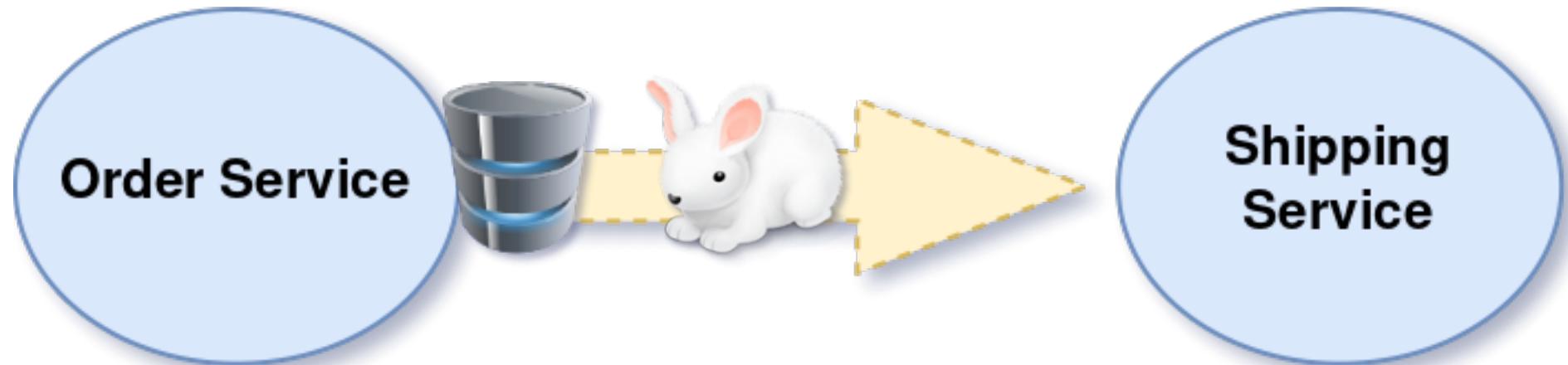
Outbox Pattern



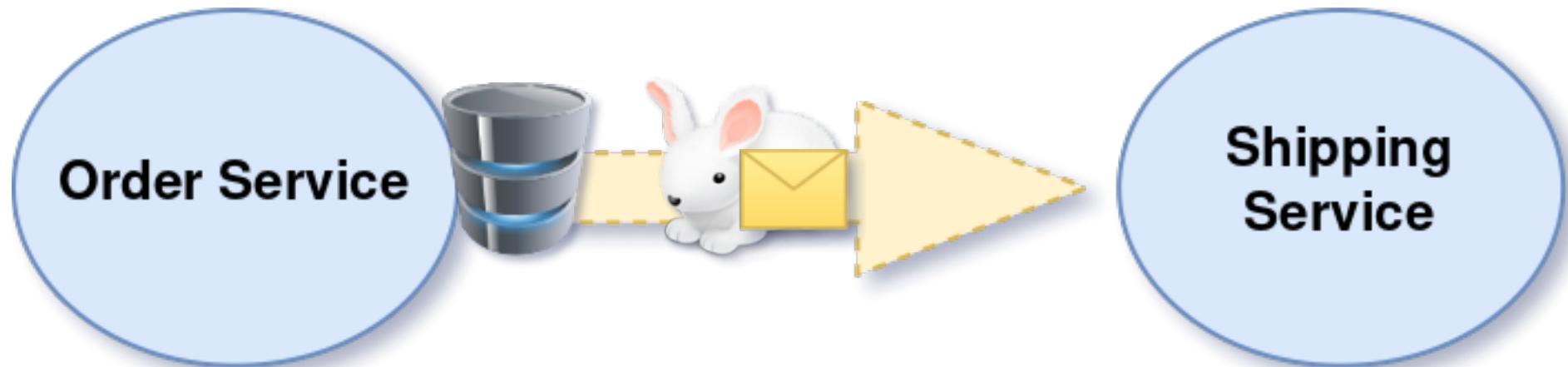
Outbox Pattern



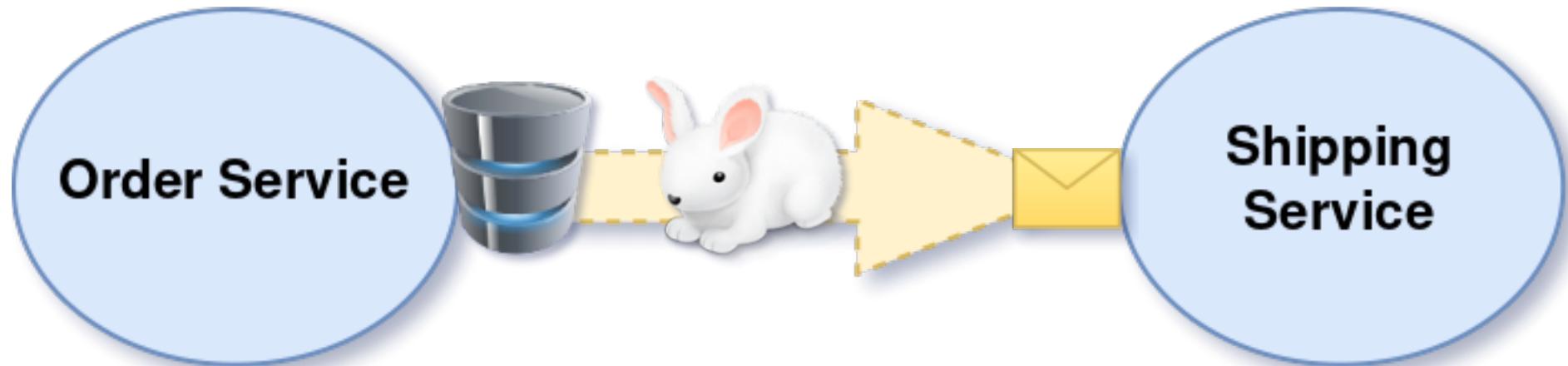
Outbox Pattern



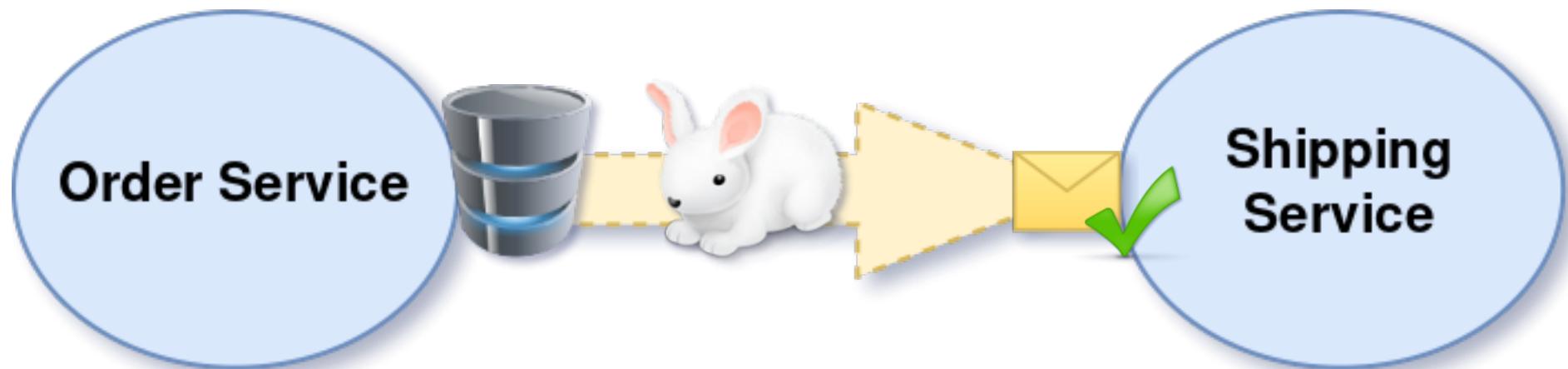
Outbox Pattern



Outbox Pattern



Outbox Pattern



Keep it within transaction

```
# [CommandHandler]
public function placeOrder(PlaceOrder $command): void
{
    $order = Order::create(
        $command->userId, $command->shippingAddress,
        $this->getProductDetails($command), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Publish event indicating that Order Was Placed */
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
}
```

Keep it within transaction

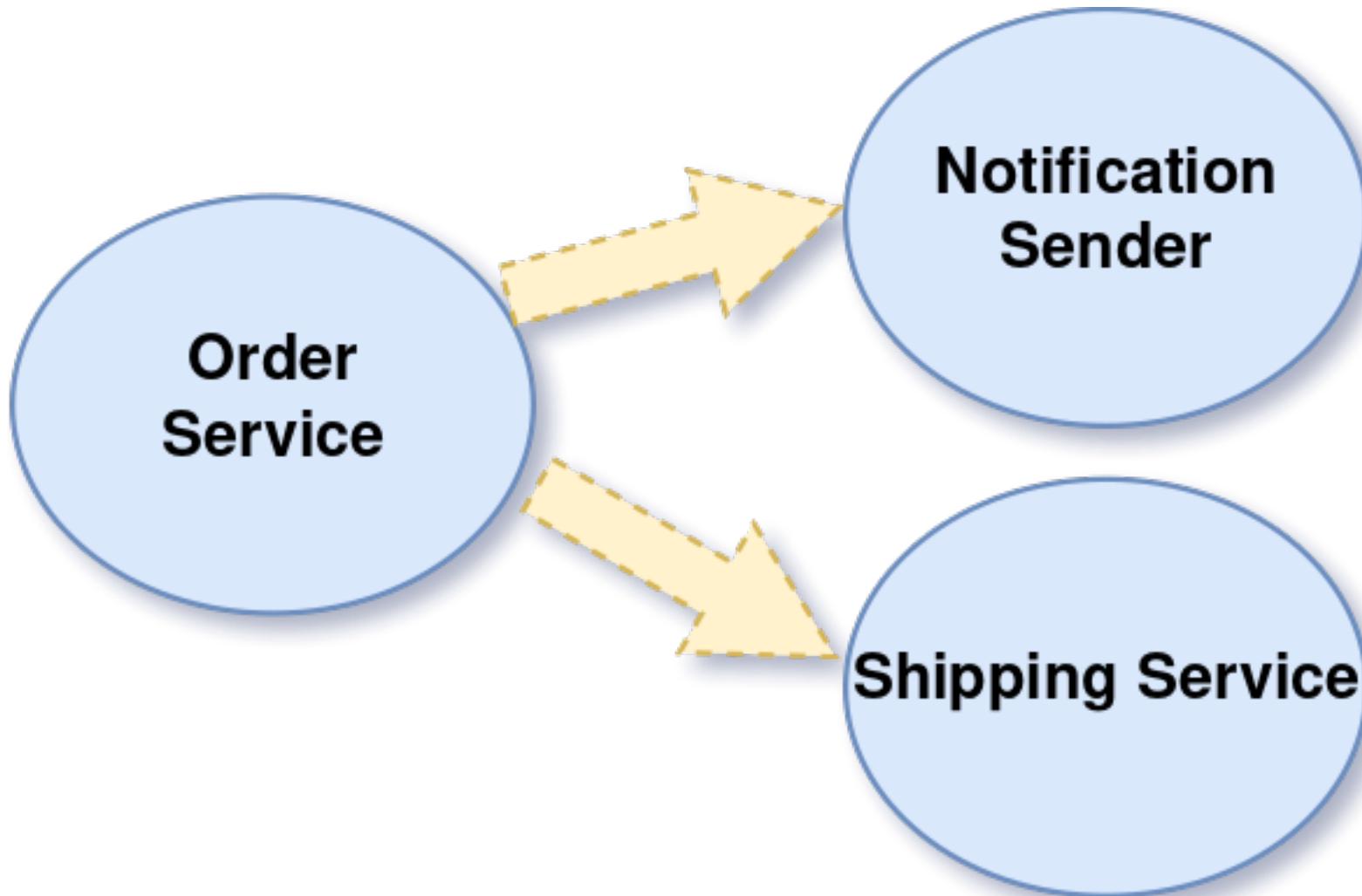
```
# [CommandHandler]  
public function placeOrder(PlaceOrder $command): void  
{  
    $order = Order::create(  
        $command->userId, $command->shippingAddress,  
        $this->getProductDetails($command), $this->clock  
    );  
    /** Storing order in database */  
    $this->orderRepository->save($order);  
  
    /** Publish event indicating that Order Was Placed */  
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));  
}
```

Keep it within transaction

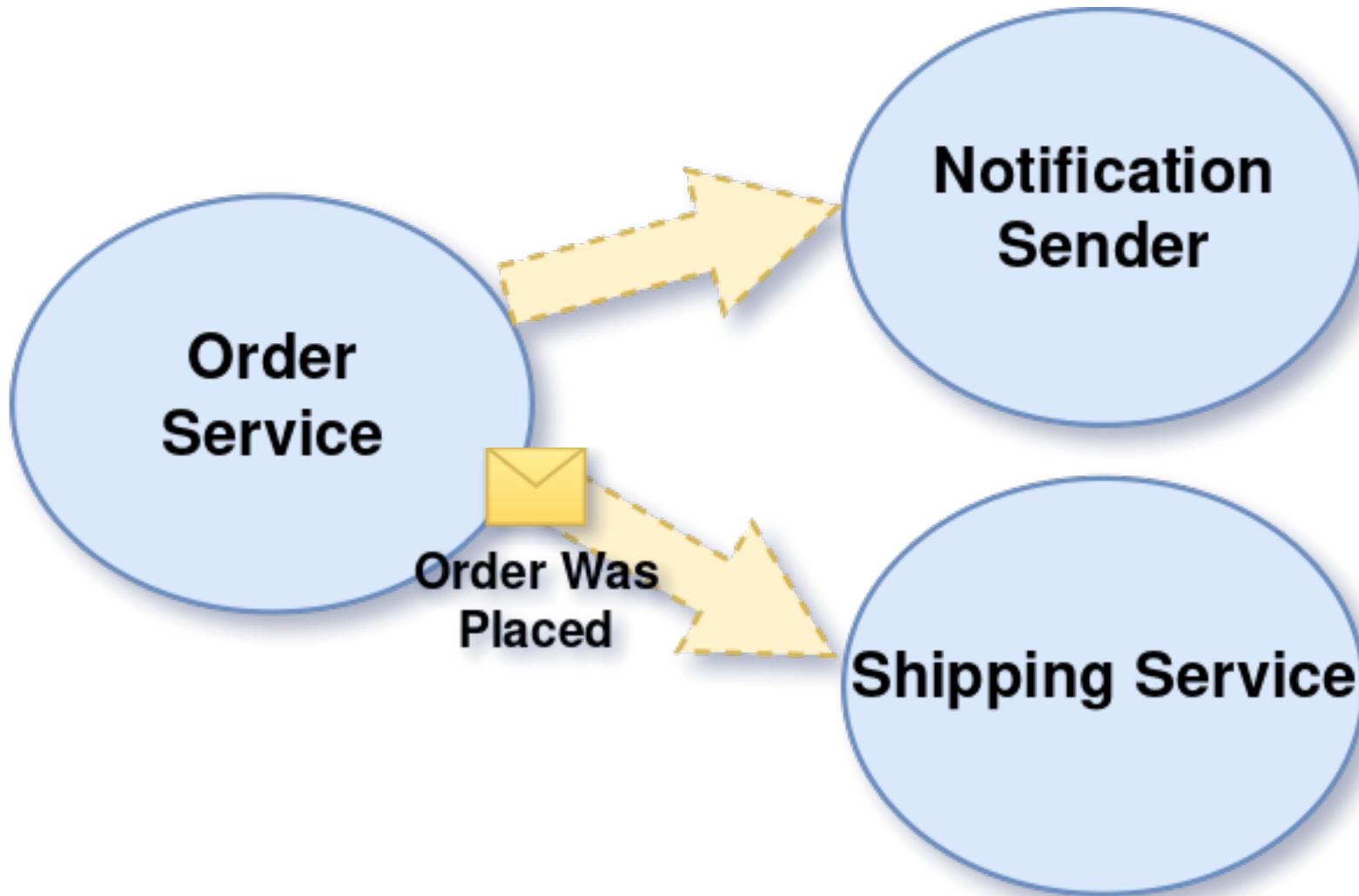
```
# [CommandHandler]
public function placeOrder(PlaceOrder $command): void
{
    $order = Order::create(
        $command->userId, $command->shippingAddress,
        $this->getProductDetails($command), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Publish event indicating that Order Was Placed */
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
}
```

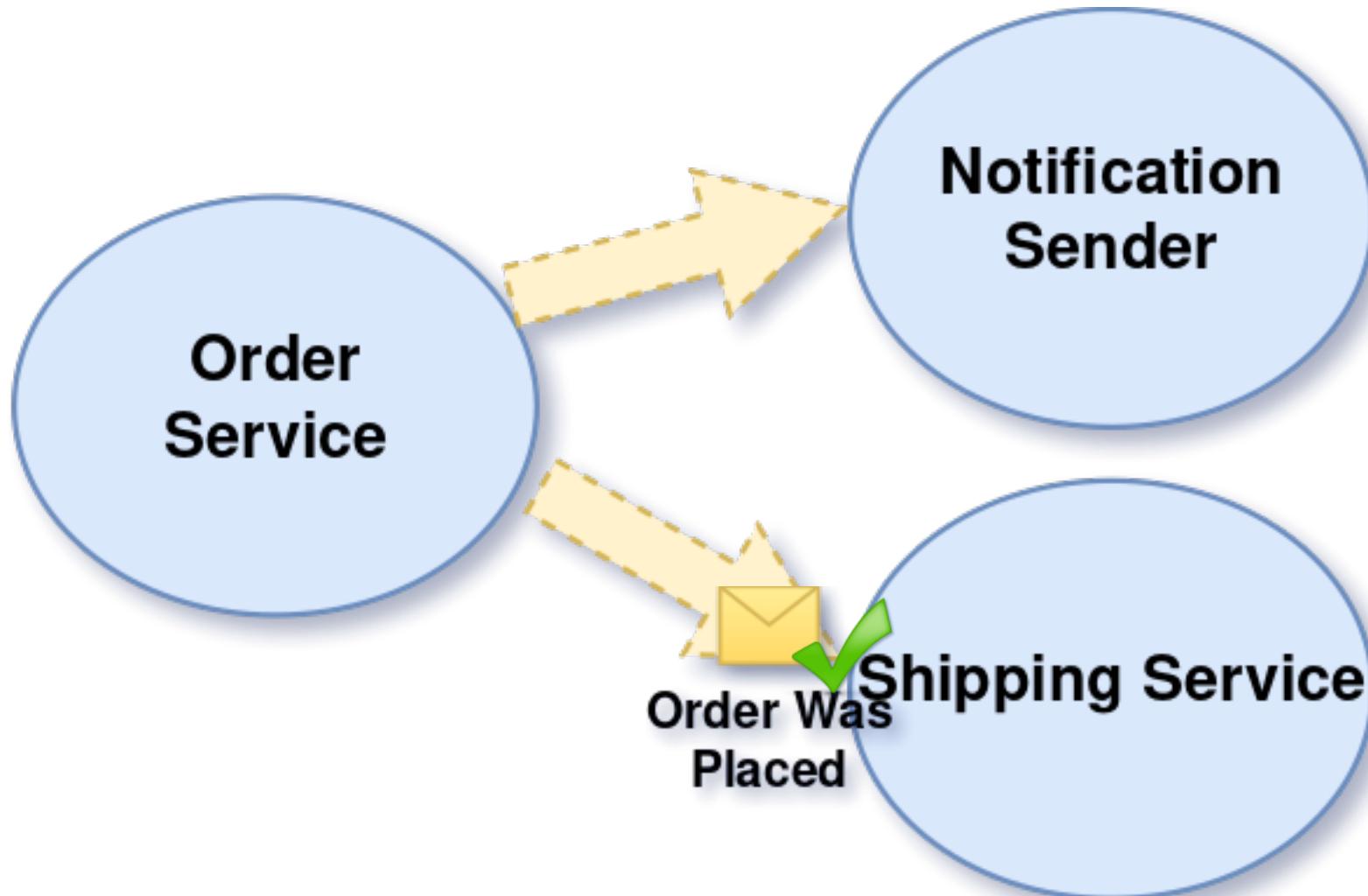
Handle each message in full isolation



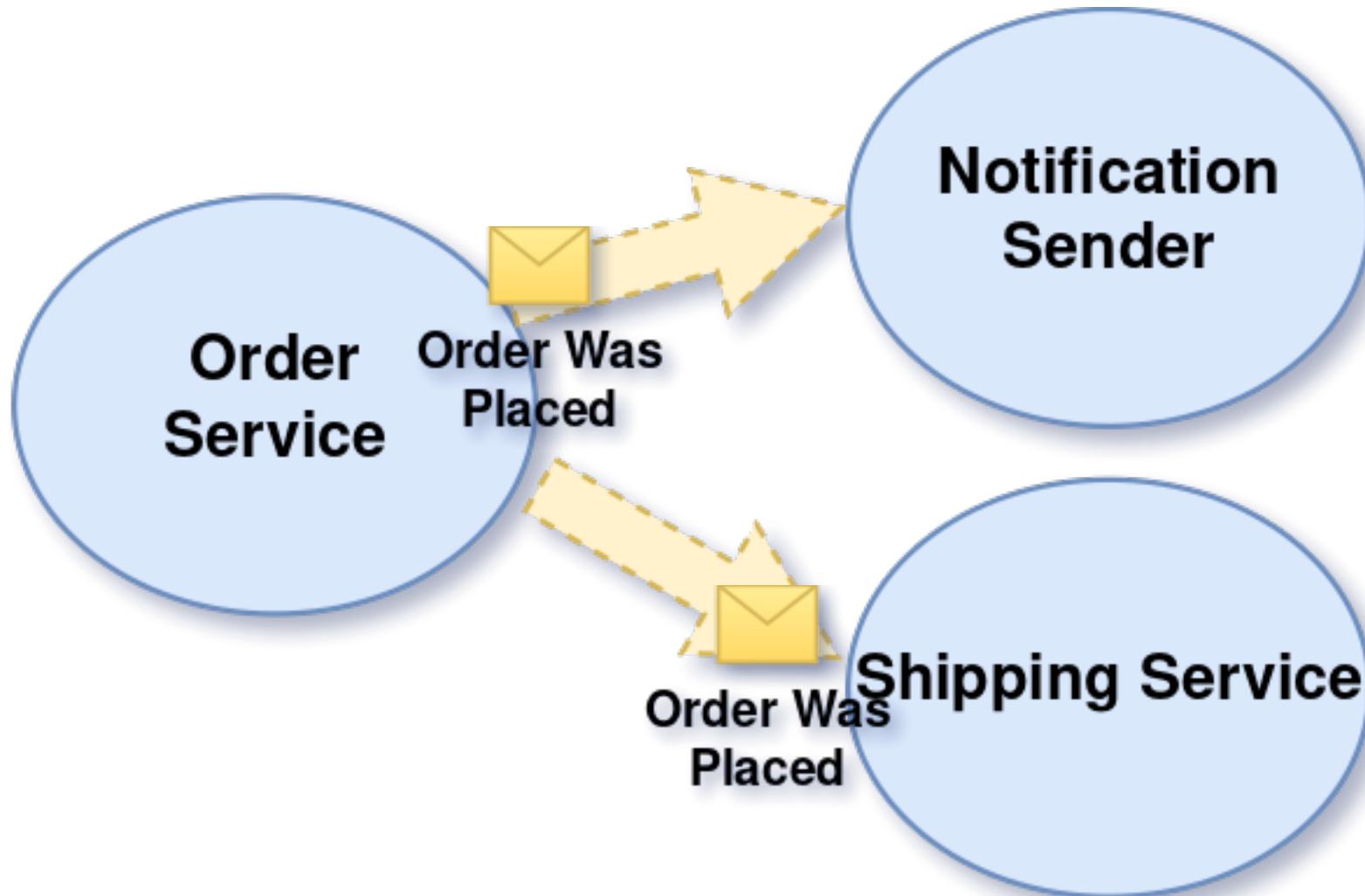
Handle each message in full isolation



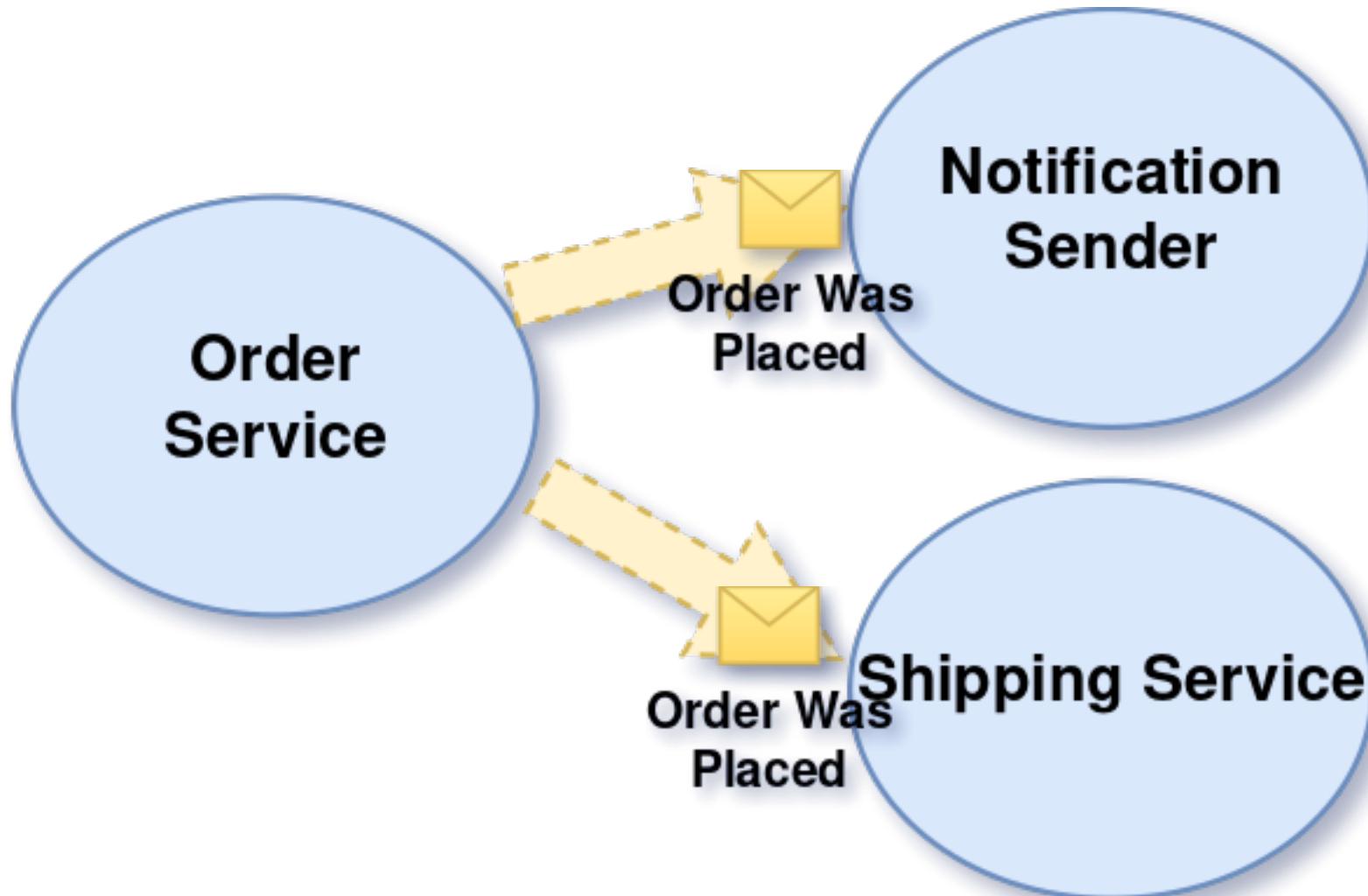
Handle each message in full isolation



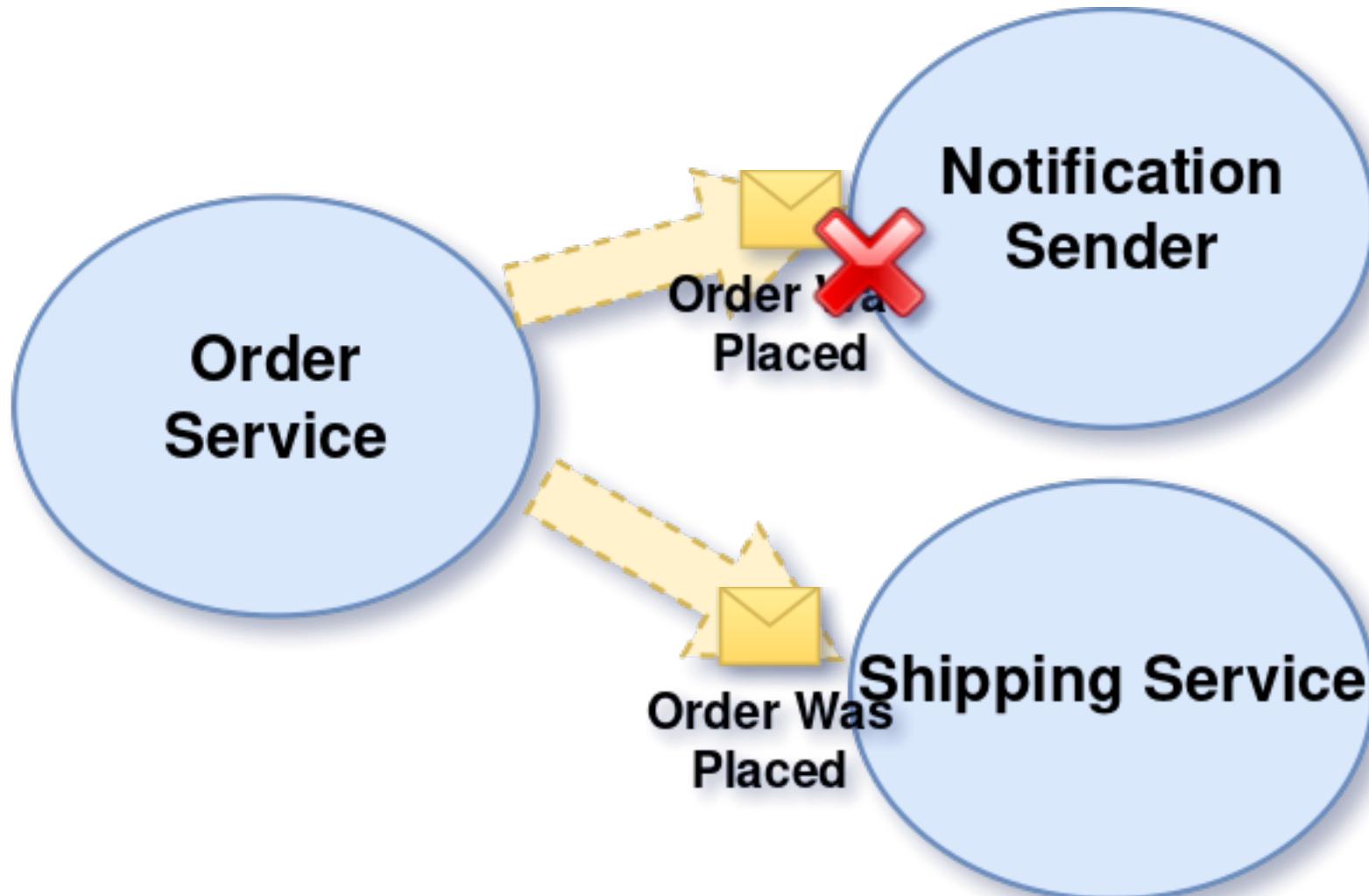
Handle each message in full isolation



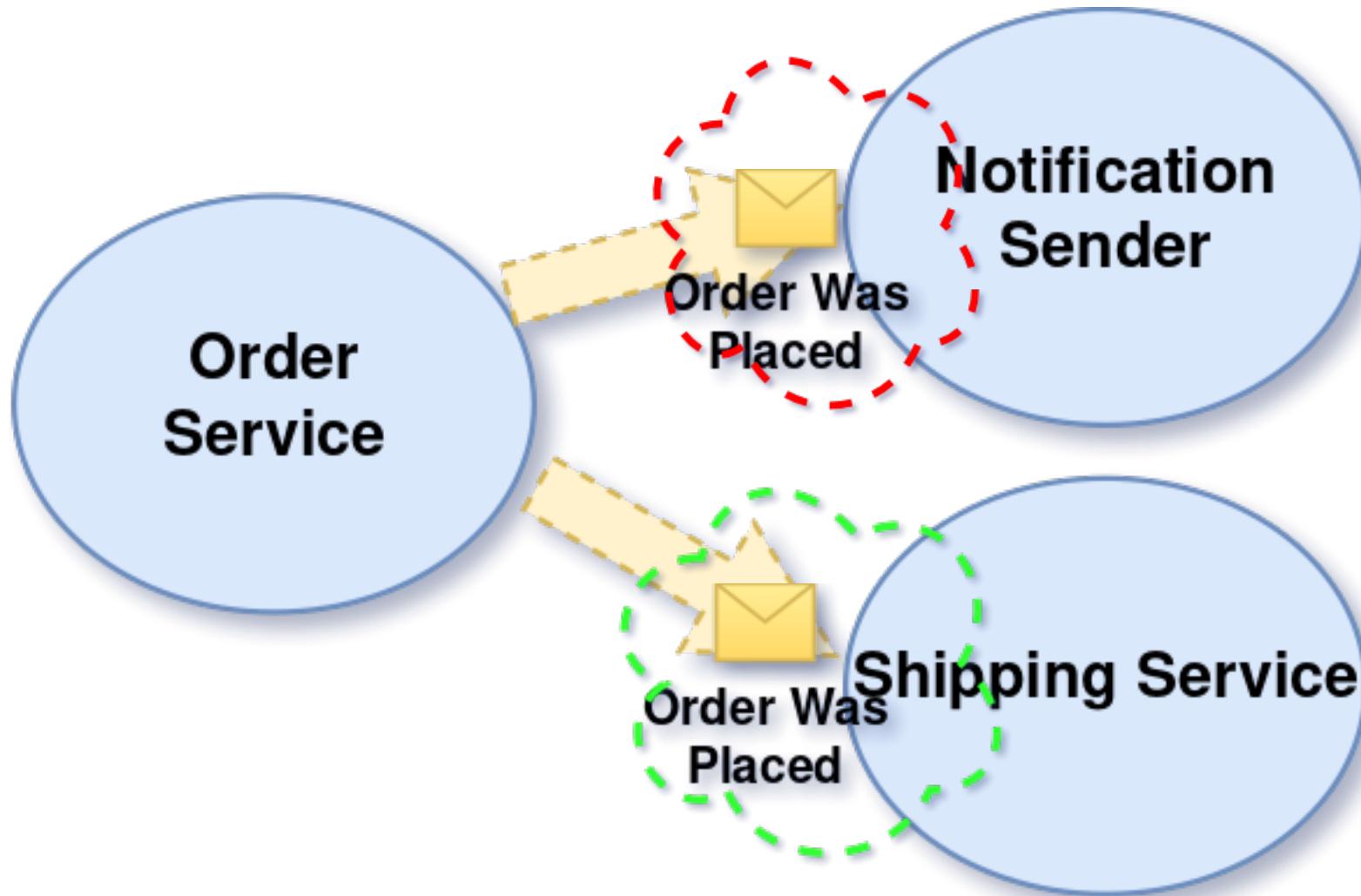
Handle each message in full isolation



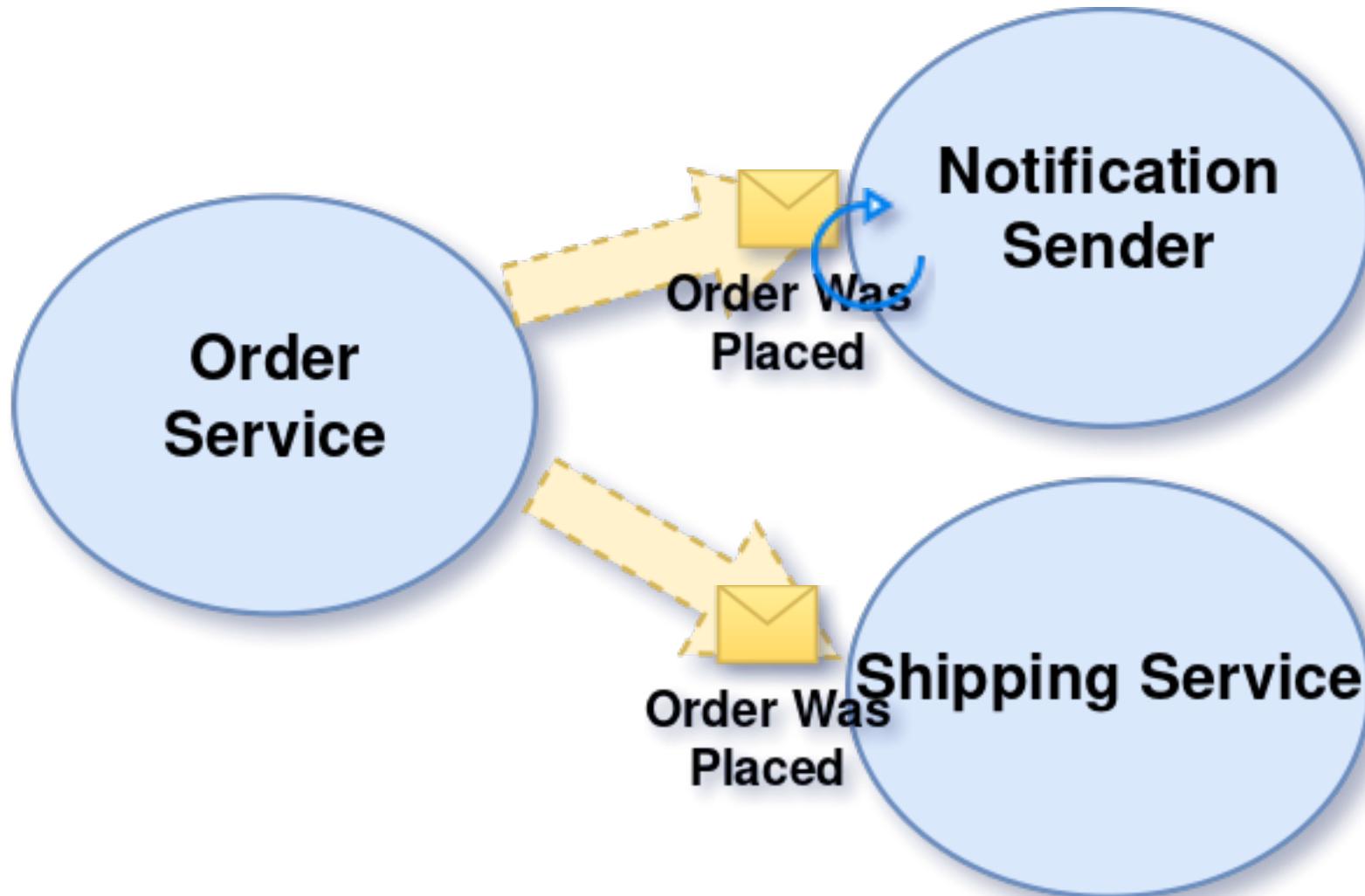
Handle each message in full isolation



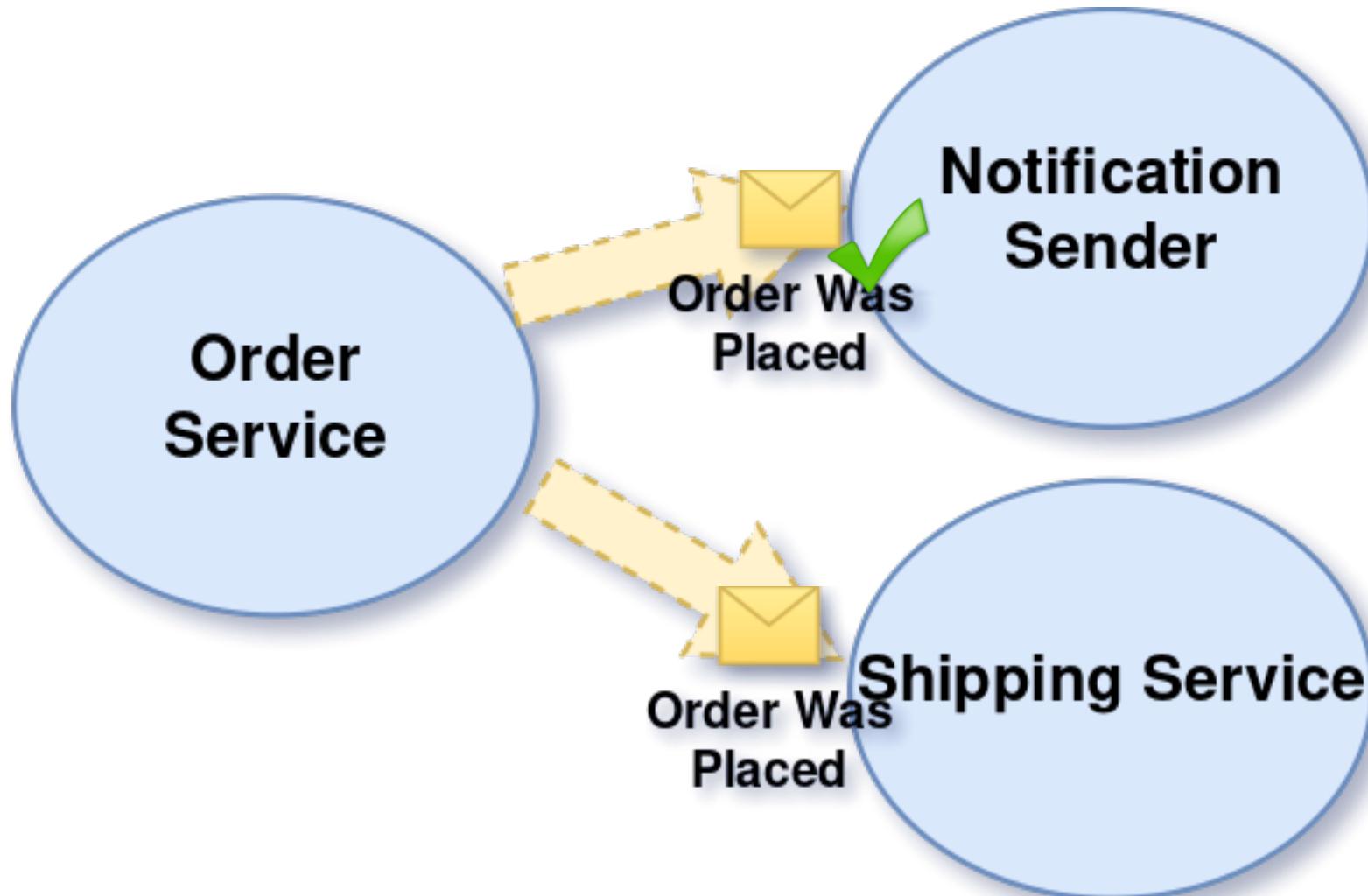
Handle each message in full isolation



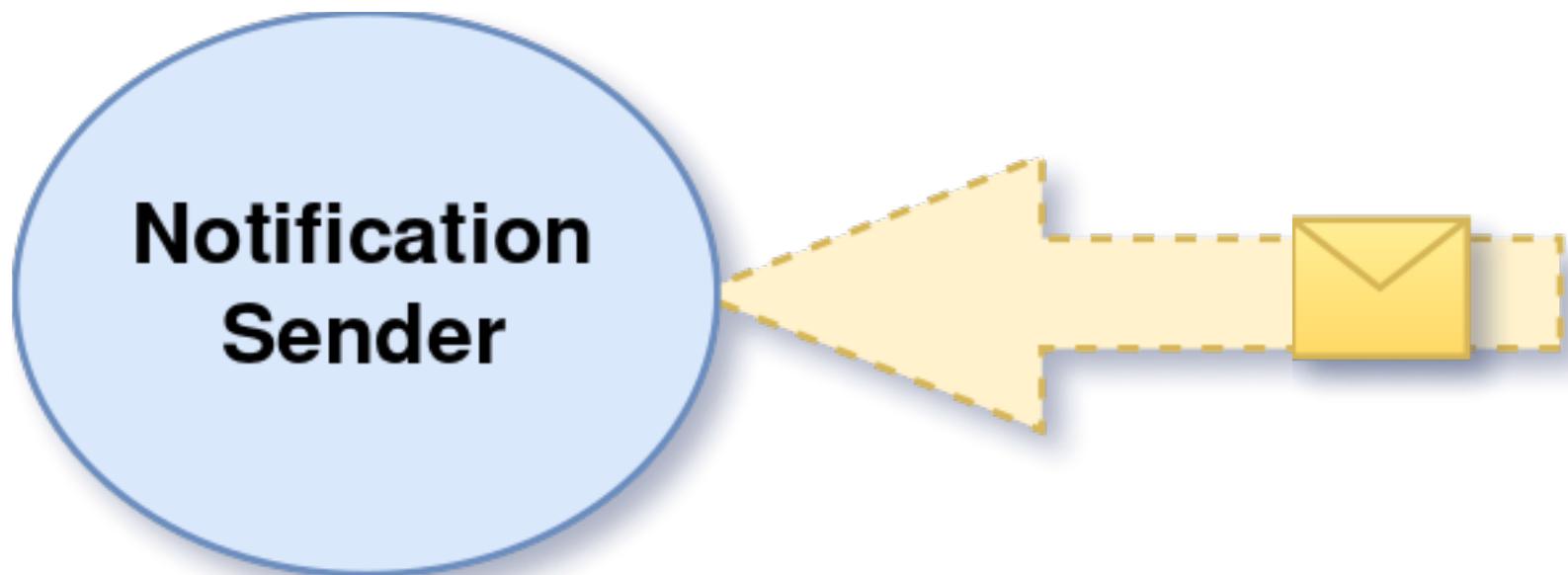
Handle each message in full isolation



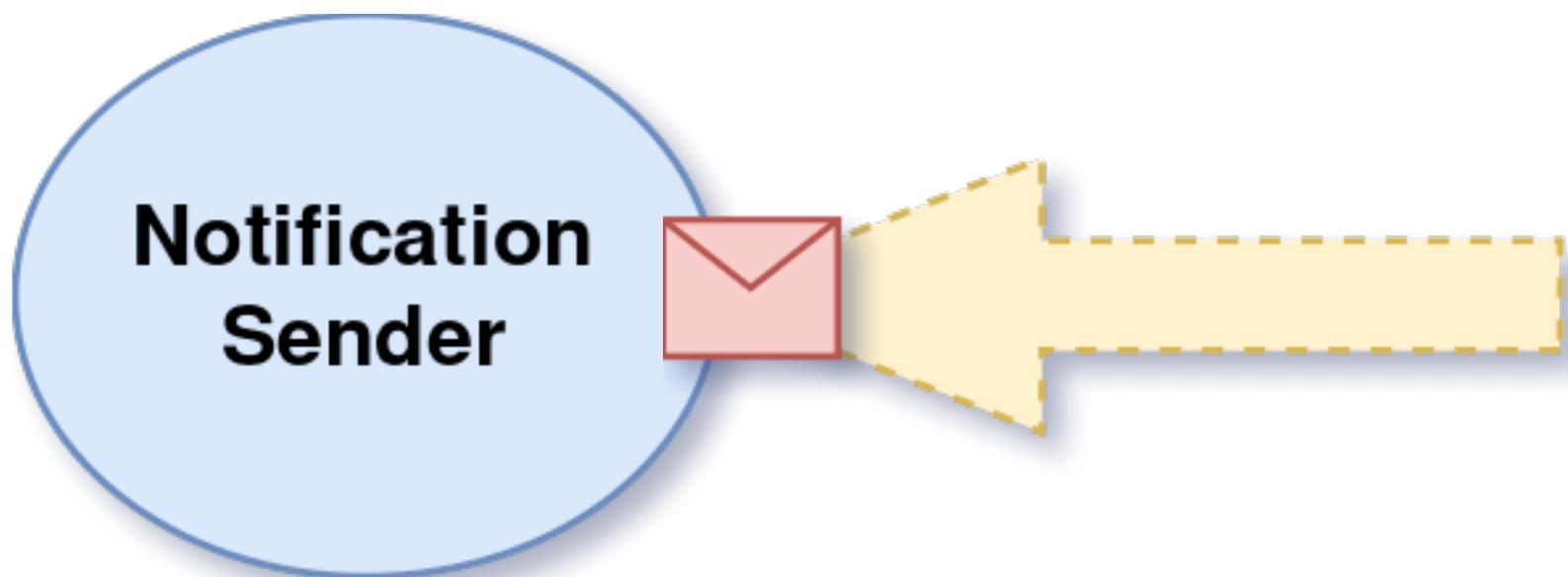
Handle each message in full isolation



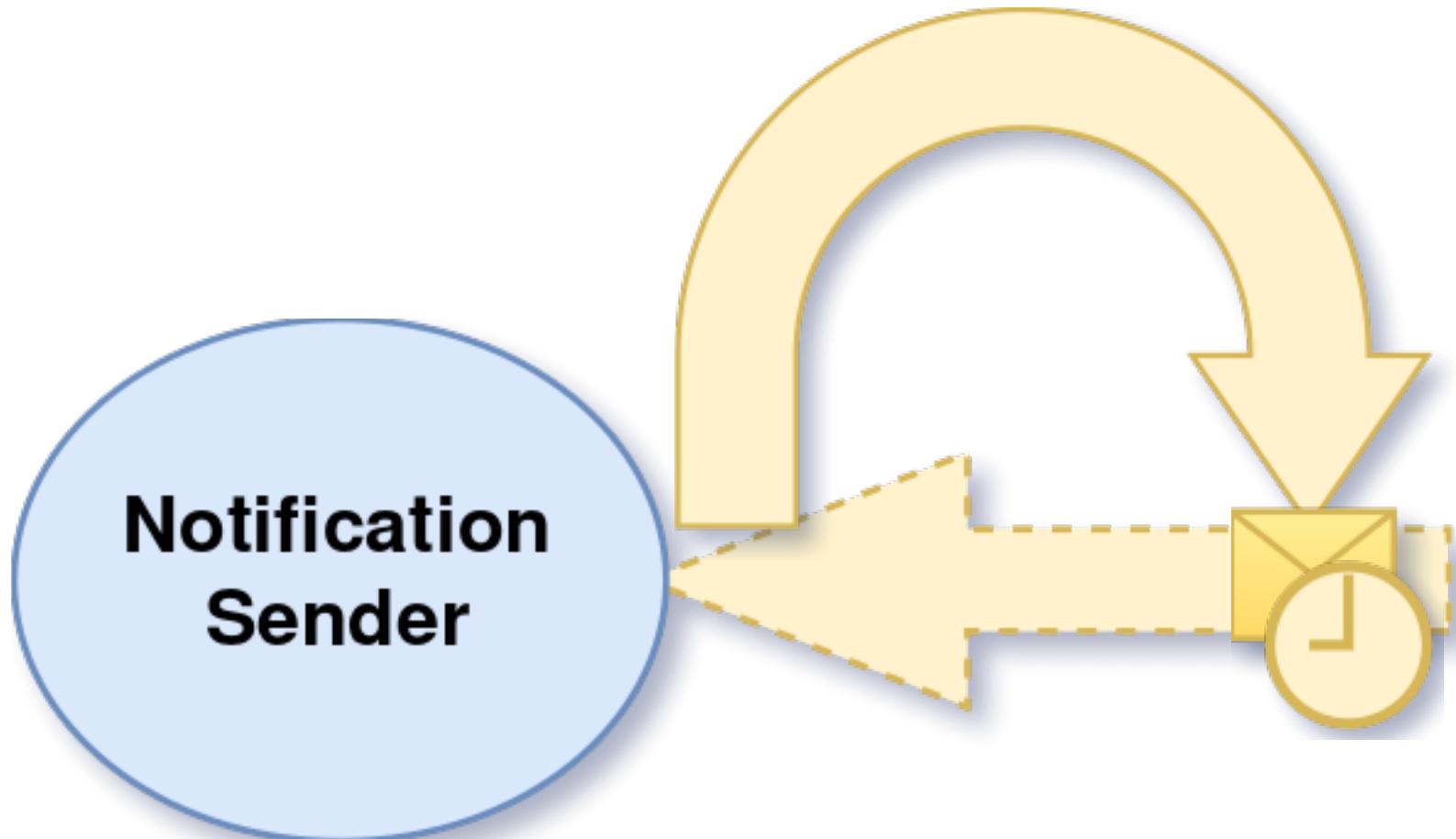
Let it self-heal



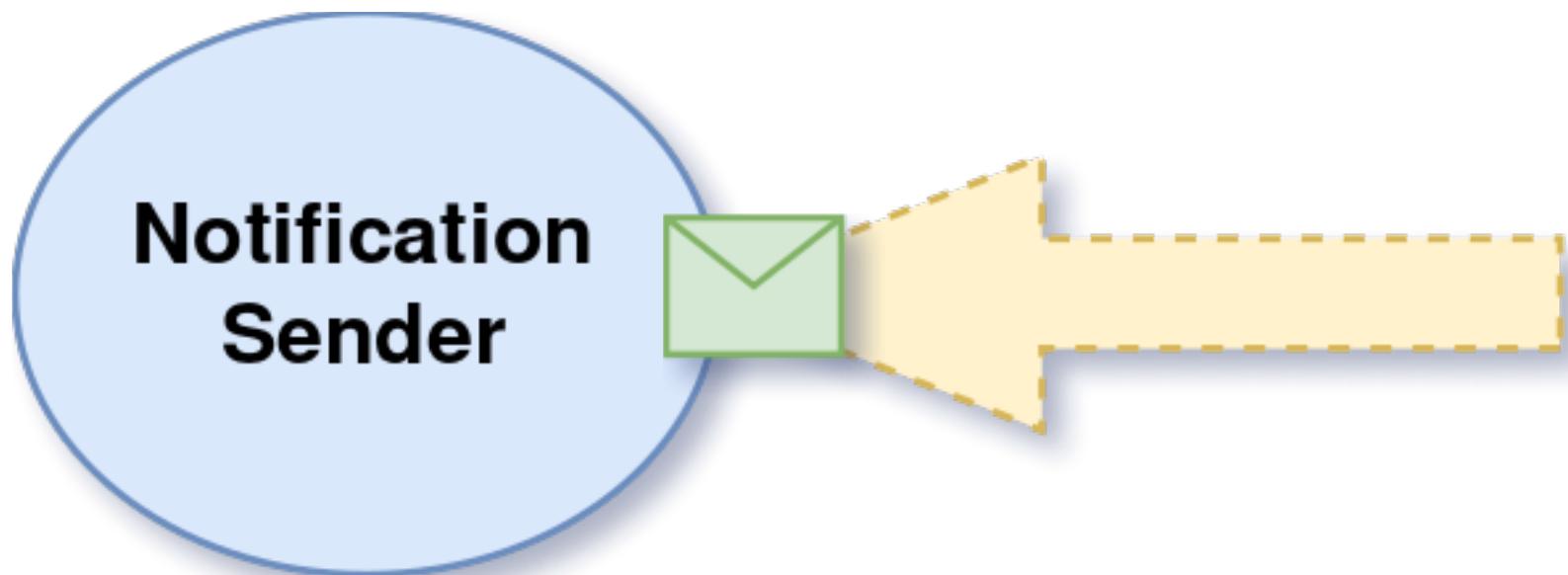
Let it self-heal



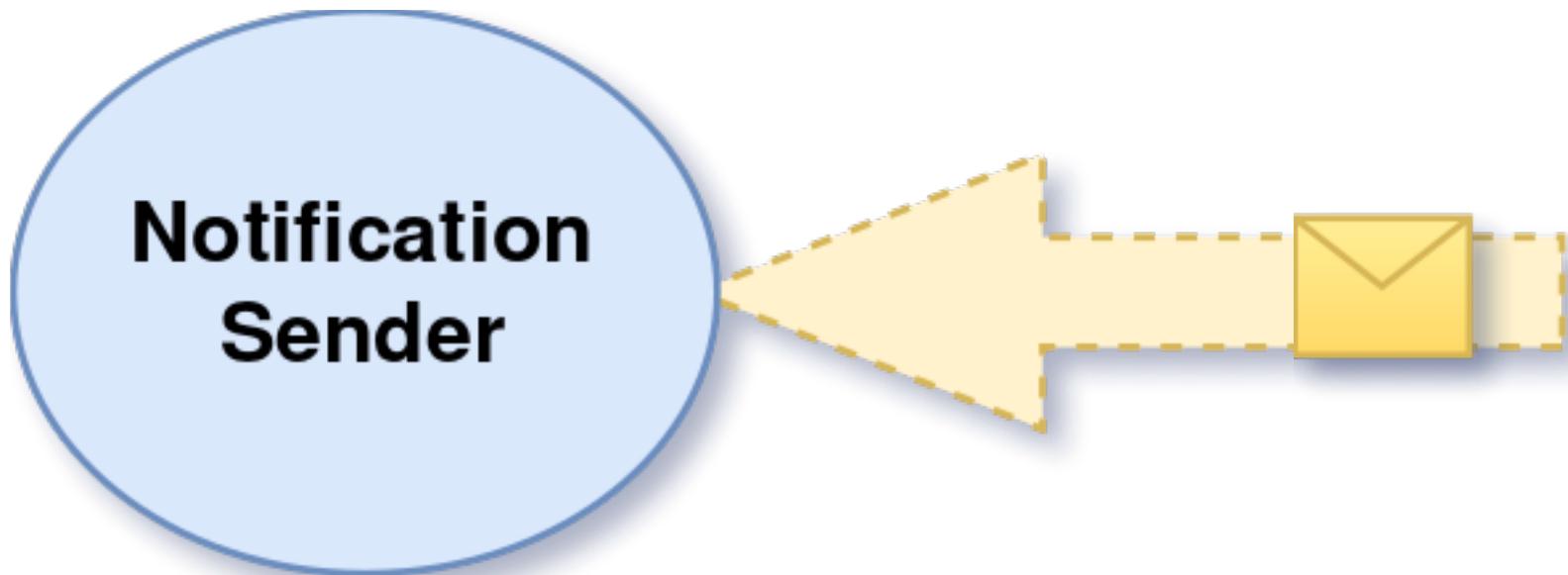
Let it self-heal



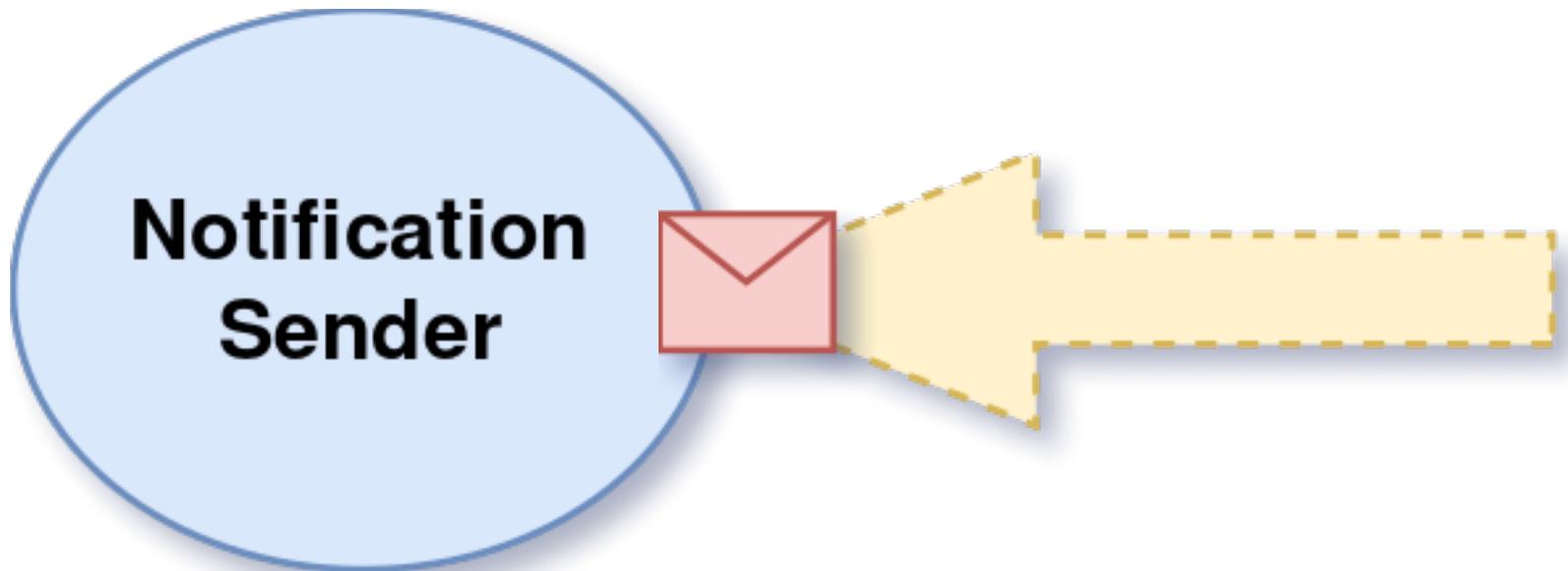
Let it self-heal



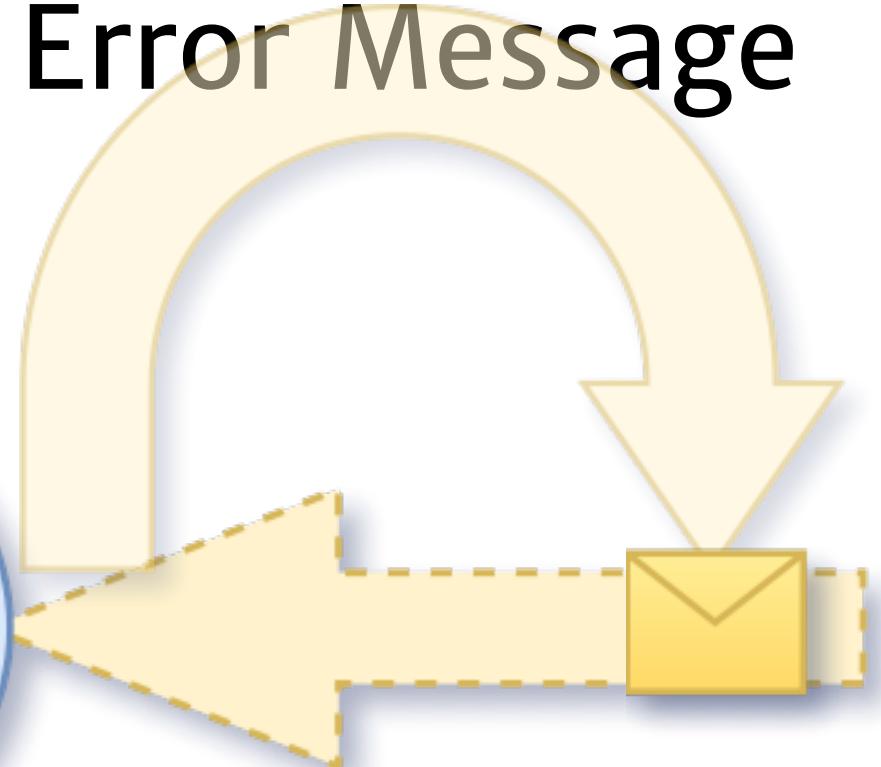
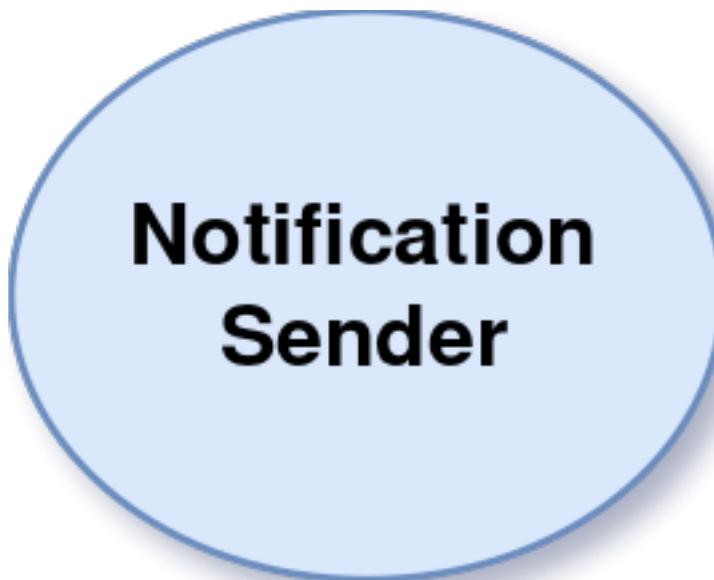
Secure the Error Message



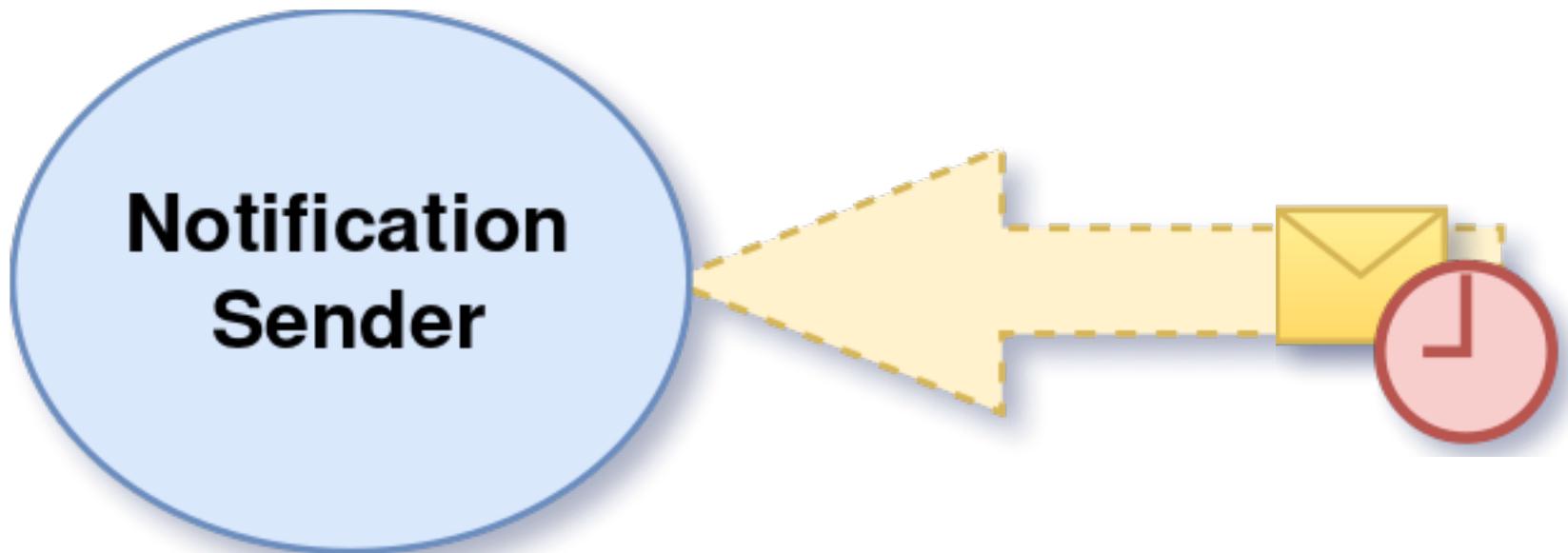
Secure the Error Message



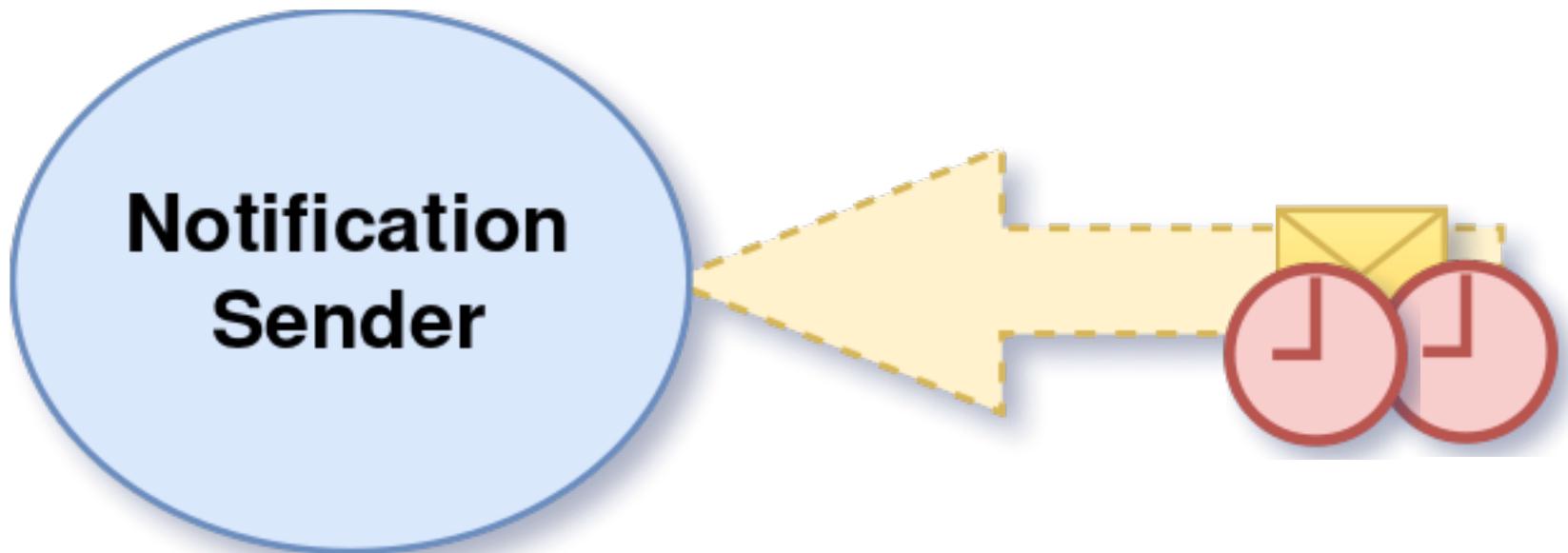
Secure the Error Message



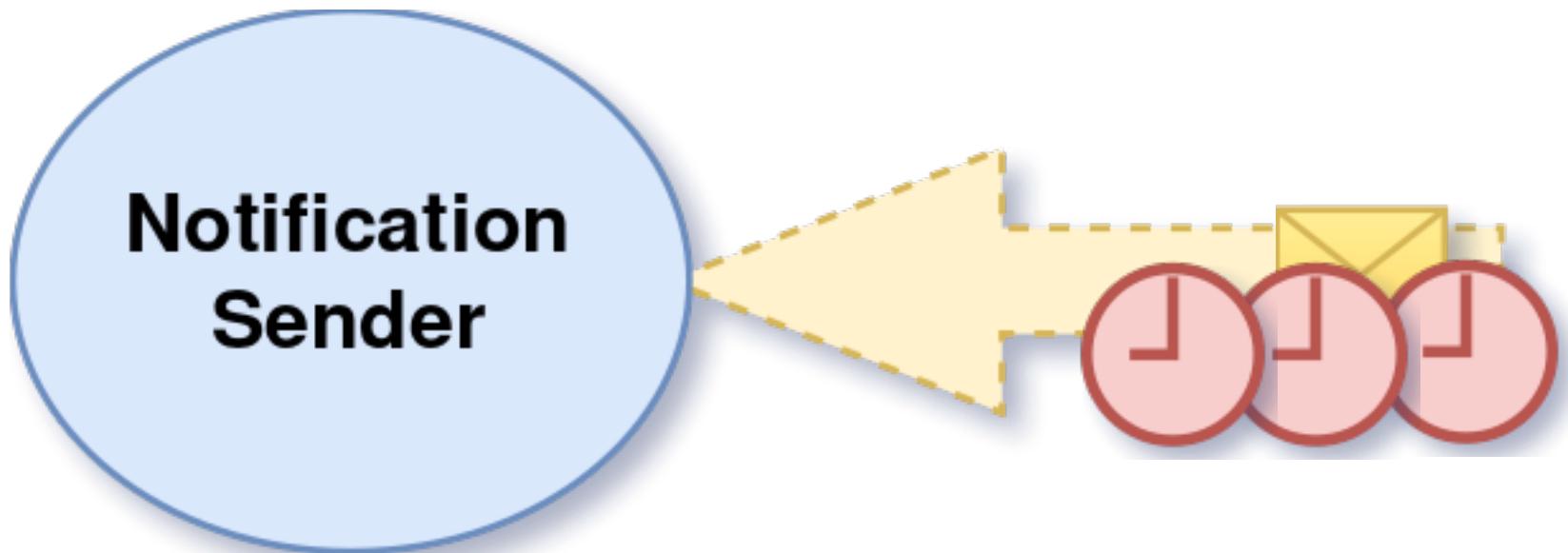
Secure the Error Message



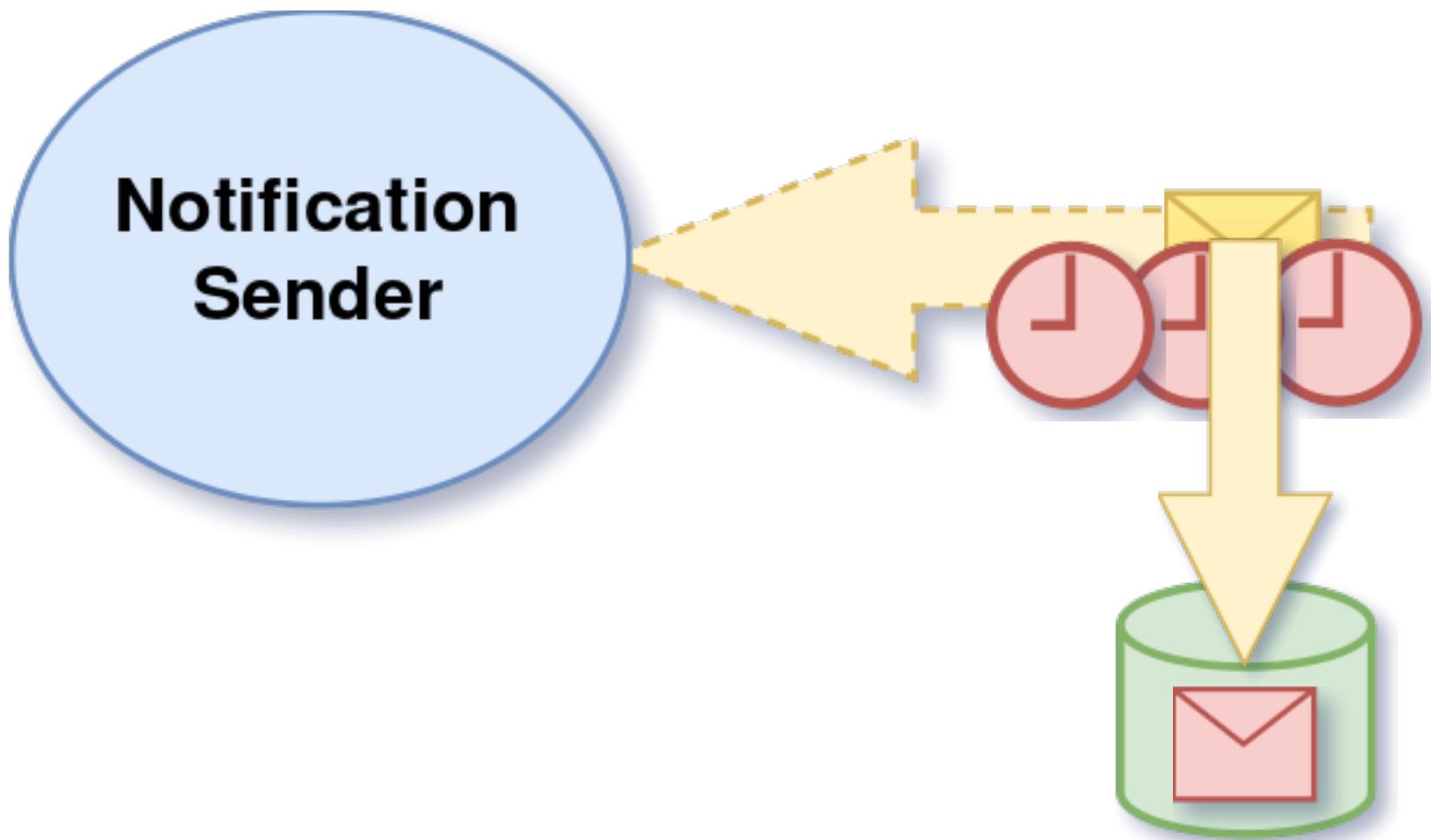
Secure the Error Message



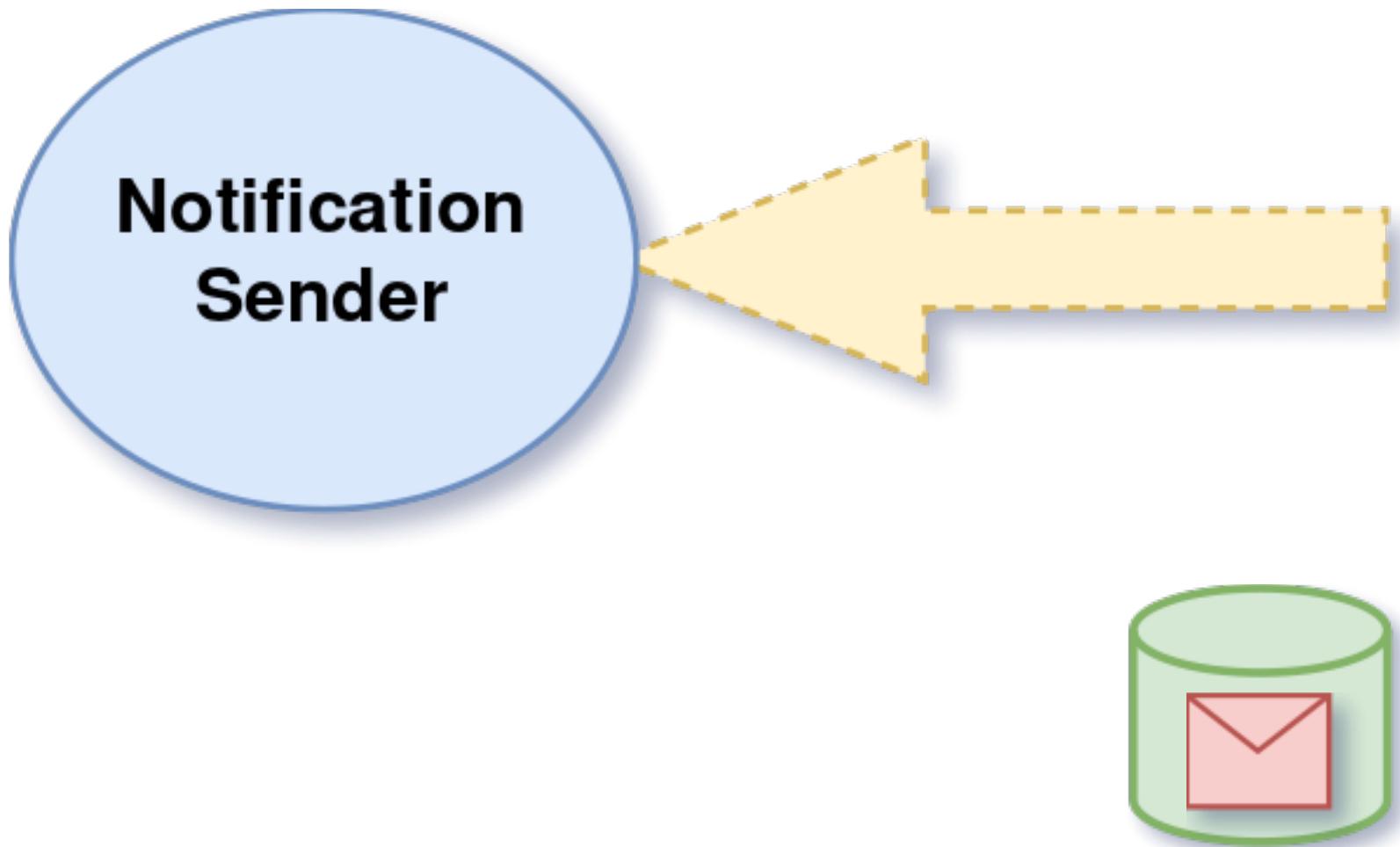
Secure the Error Message



Secure the Error Message



Recover Business Flow



Recover Business Flow

Available commands for the "ecotone:deadletter" namespace:

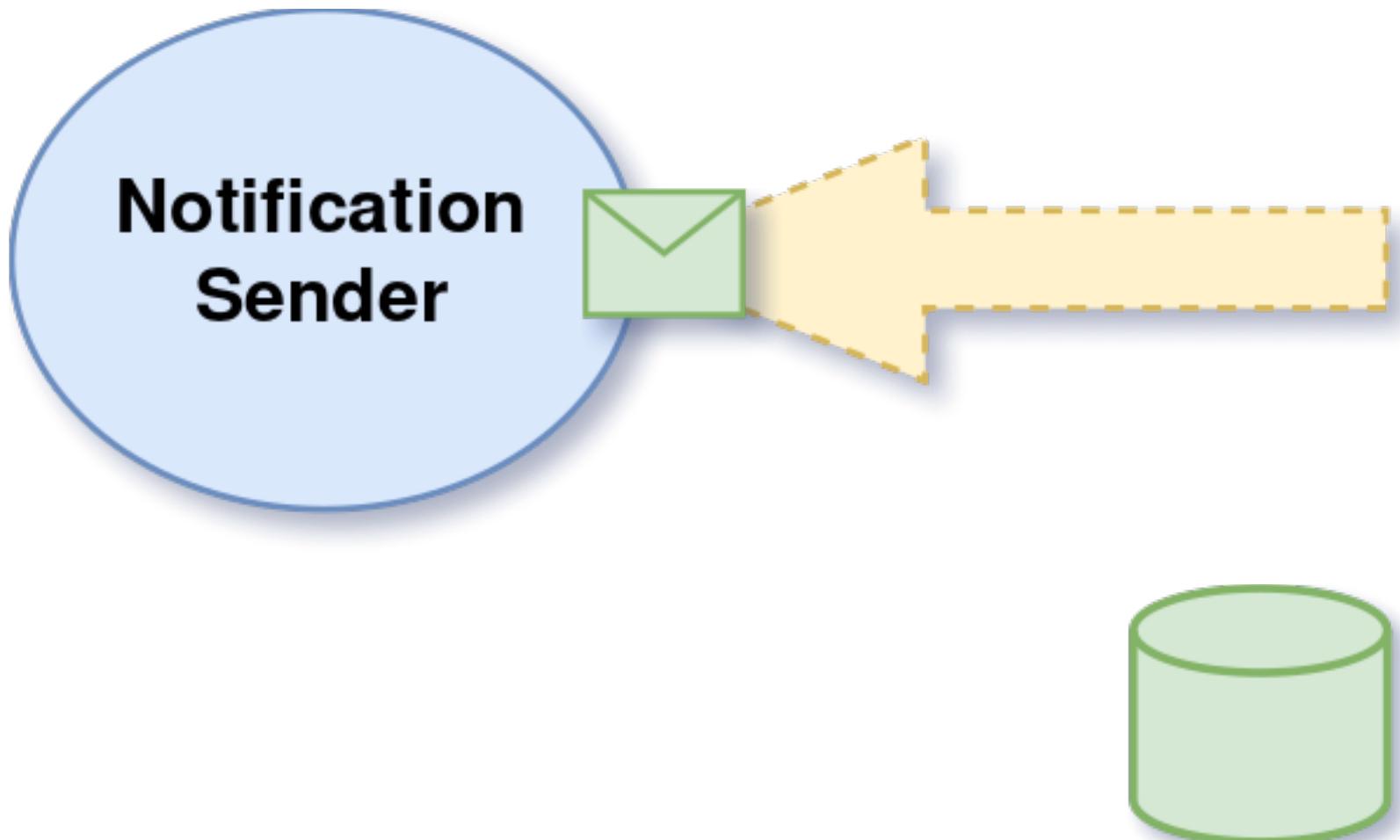
```
ecotone:deadletter:delete
ecotone:deadletter:help
ecotone:deadletter:list
ecotone:deadletter:replay
ecotone:deadletter:replayAll
ecotone:deadletter:show
```



Recover Business Flow



Recover Business Flow



Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

Dashboard

Your Services

customer_service

backoffice_service



Service Name	Errors	Actions
customer_service	2	Get more details
backoffice_service	0	Get more details

Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
9e6bc785-5c3c-4da6-a721-fb7a909d7ce8	Can't render template, missing issue id parameter	Show	01-03-2023 20:56:56.000000	<button>Replay</button> <button>Delete</button>
6e22400e-6c3b-47ec-b109-1500d8ca375c	Can't render template, missing issue id parameter	Show	01-03-2023 20:42:59.000000	<button>Replay</button> <button>Delete</button>

Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
9e6bc785-5c3c-4da6-a721-fb7a909d7ce8	Can't render template, missing issue id parameter	Show	01-03-2023 20:56:56.000000	<button>Replay</button> <button>Delete</button>
6e22400e-6c3b-47ec-b109-1500d8ca375c	Can't render template, missing issue id parameter	Show	01-03-2023 20:42:59.000000	<button>Replay</button> <button>Delete</button>



Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

CU	Ref	Created At	Actions
	1	2023-06-05 14:56:00	<button>Replay</button> <button>Delete</button>
	2	2023-06-05 14:59:00	<button>Replay</button> <button>Delete</button>

```
#0 [internal function]: App\Models\IssueSubscriber->sendNotificationToConfirmReceivedIssue() #1
/data/app/vendor/ecotone/ecotone/src/Messaging/Handler/Processor\MethodInvoker\MethodInvoker::call_user_func_array() #2
/data/app/vendor/ecotone/ecotone/src/Messaging/Handler/Processor\MethodInvoker\MethodInvoker::processMessage() #3
/data/app/vendor/ecotone/ecotone/src/Messaging/Handler/RequestReplyProducer\WrapWithMessageBuilder\WrapWithMessageBuilder::processMessage() #4
/data/app/vendor/ecotone/ecotone/src/Messaging/Handler/RequestReplyProducer\HandleWithReply\HandleWithReply::handleWithReply() #5
```

Close

Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
9e6bc785-5c3c-4da6-a721-fb7a909d7ce8	Can't render template, missing issue id parameter	Show	01-03-2023 20:56:56.000000	 Replay Delete
6e22400e-6c3b-47ec-b109-1500d8ca375c	Can't render template, missing issue id parameter	Show	01-03-2023 20:42:59.000000	Replay Delete

Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
9e6bc785-5c3c-4da6-a721-fb7a909d7ce8	Can't render template, missing issue id parameter	Show	01-03-2023 20:56:56.000000	<button>Replay</button> <button>Delete</button>
6e22400e-6c3b-47ec-b109-1500d8ca375c	Can't render template, missing issue id parameter	Show	01-03-2023 20:42:59.000000	<button>Replay</button> <button>Delete</button>



Manage Multiple Applications

- Ecotone Pulse

Ecotone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
6e22400e-6c3b-47ec-b109-1500d8ca375c	Can't render template, missing issue id parameter	Show	01-03-2023 20:42:59.000000	<button>Replay</button> <button>Delete</button>

One more important ingredient: Tracing

One more important ingredient: Tracing



- Sharing knowledge about high level overview of the system

One more important ingredient: Tracing



- Sharing knowledge about high level overview of the system



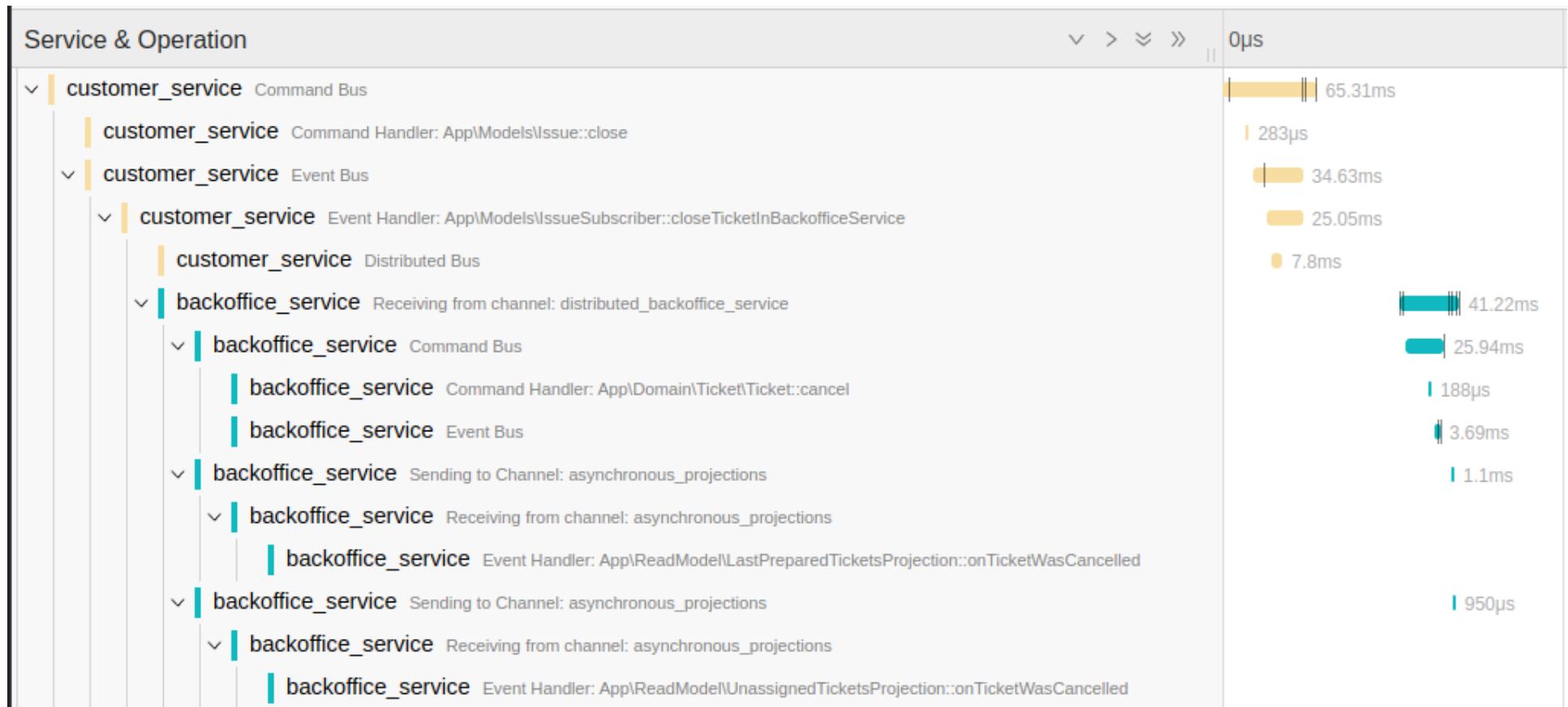
- Tracing which point of the flow had stopped or failed

One more important ingredient: Tracing

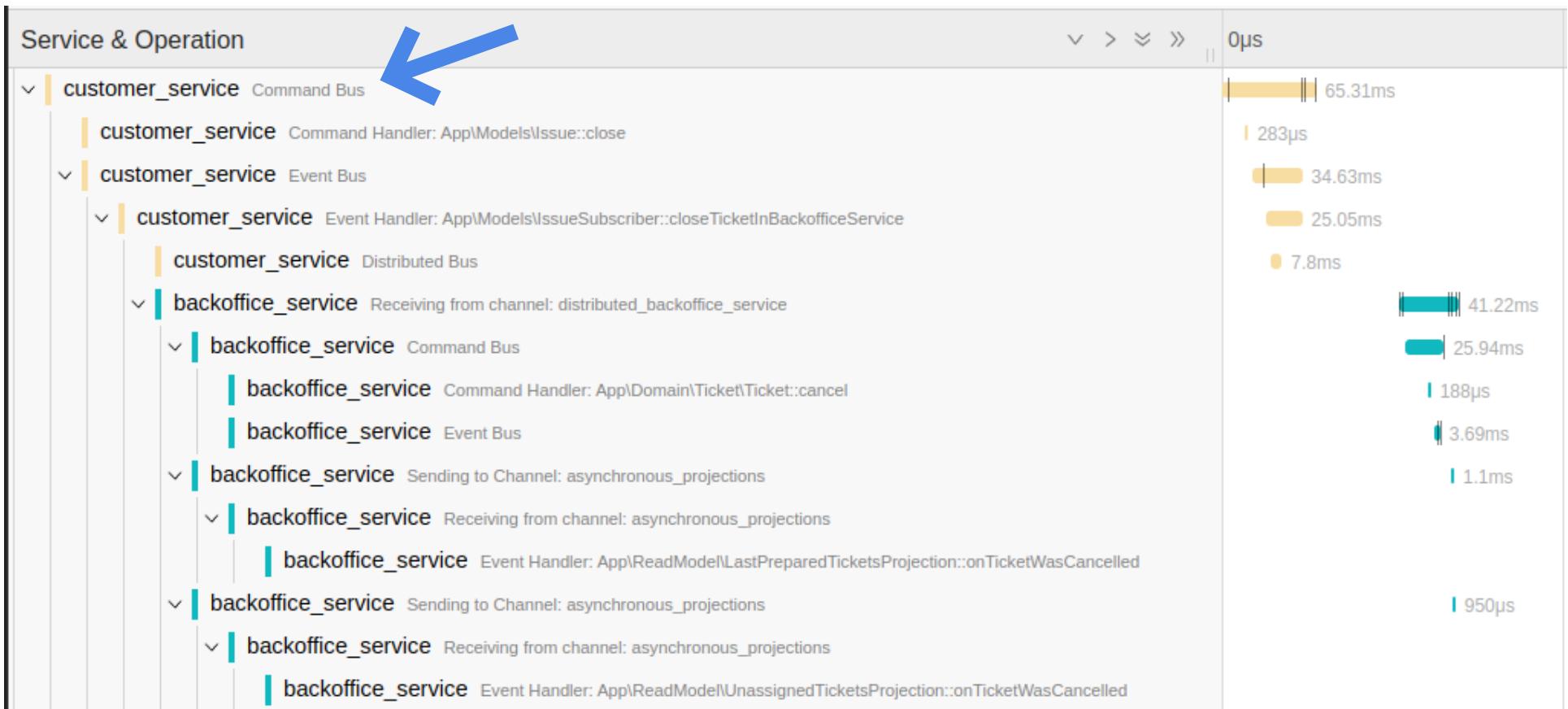


- Sharing knowledge about high level overview of the system
- Tracing which point of the flow had stopped or failed
- What is the performance at each state of the flow

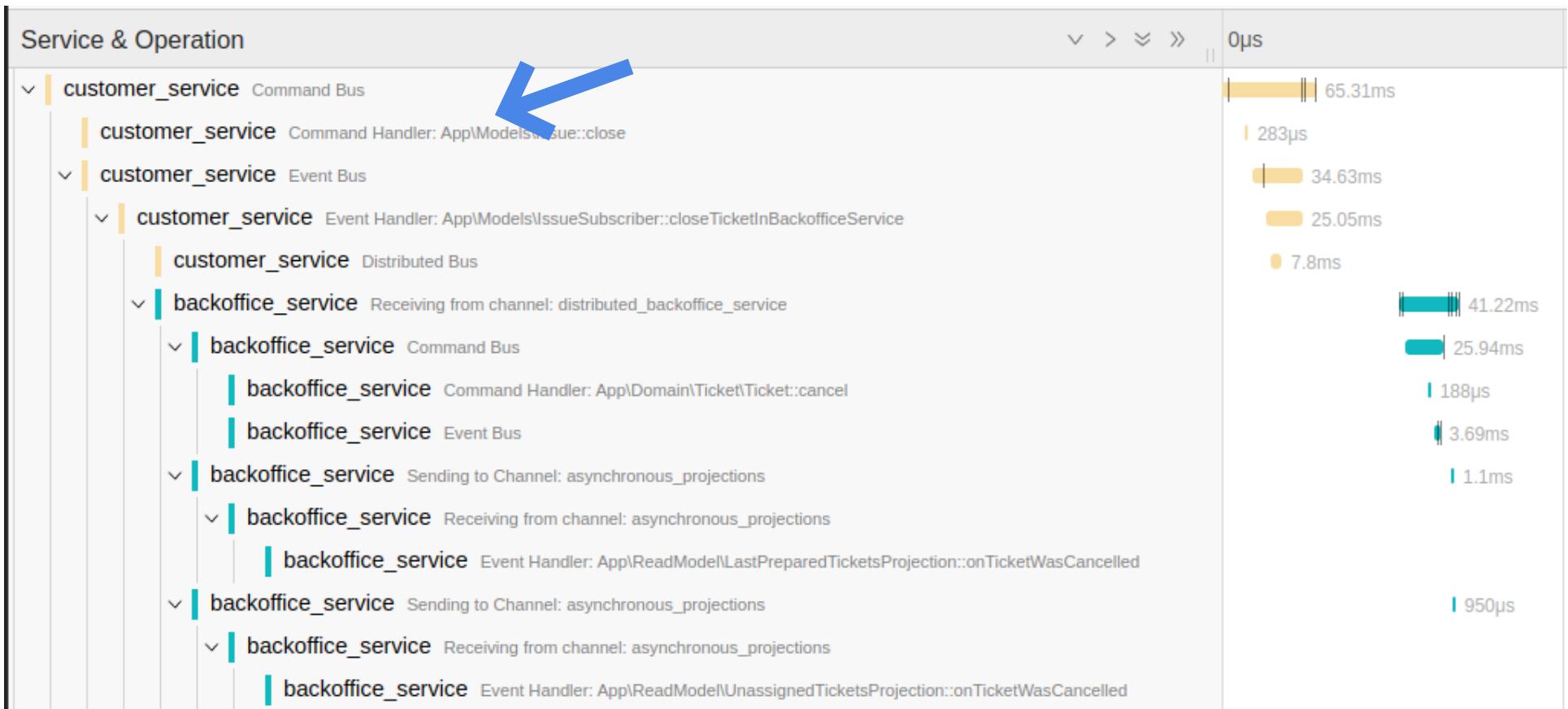
One more important ingredient: Tracing



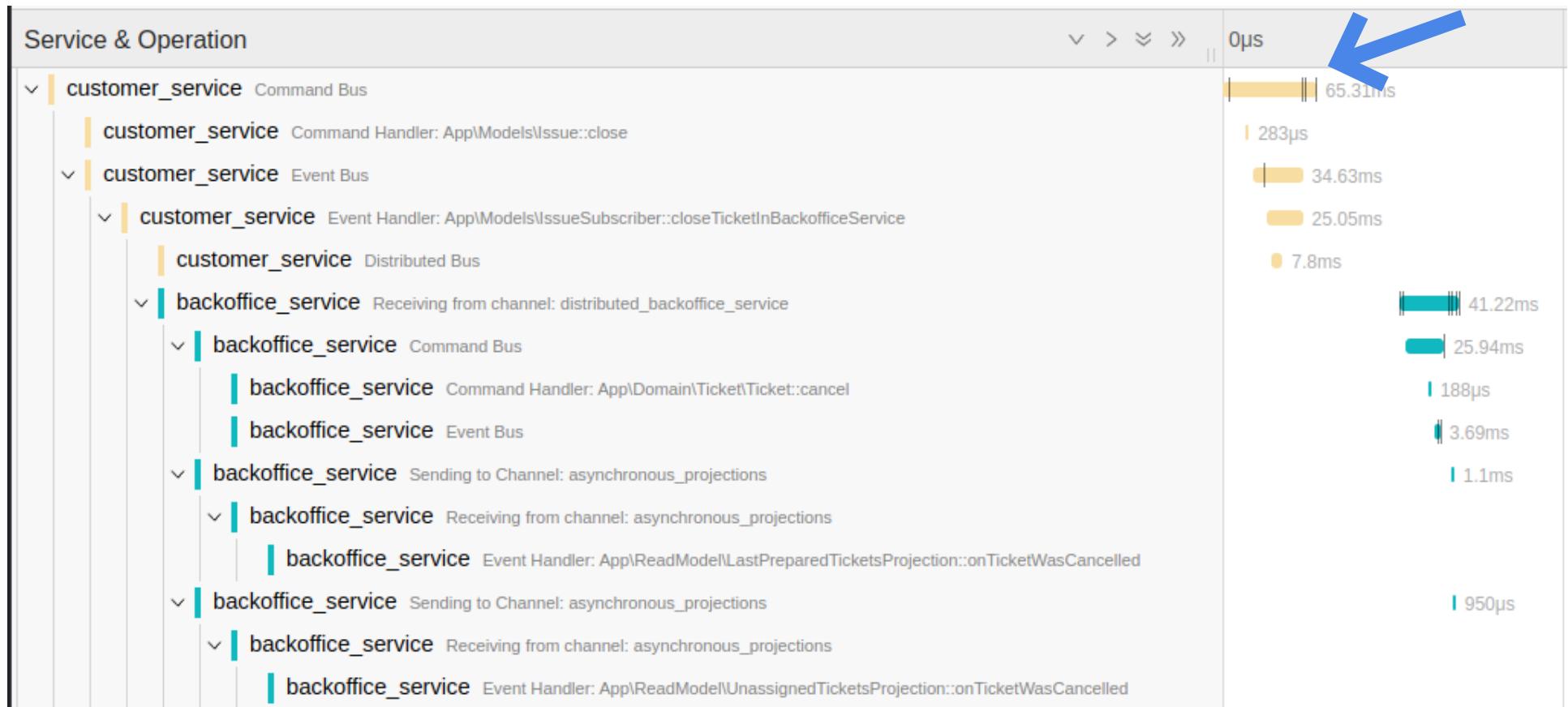
One more important ingredient: Tracing



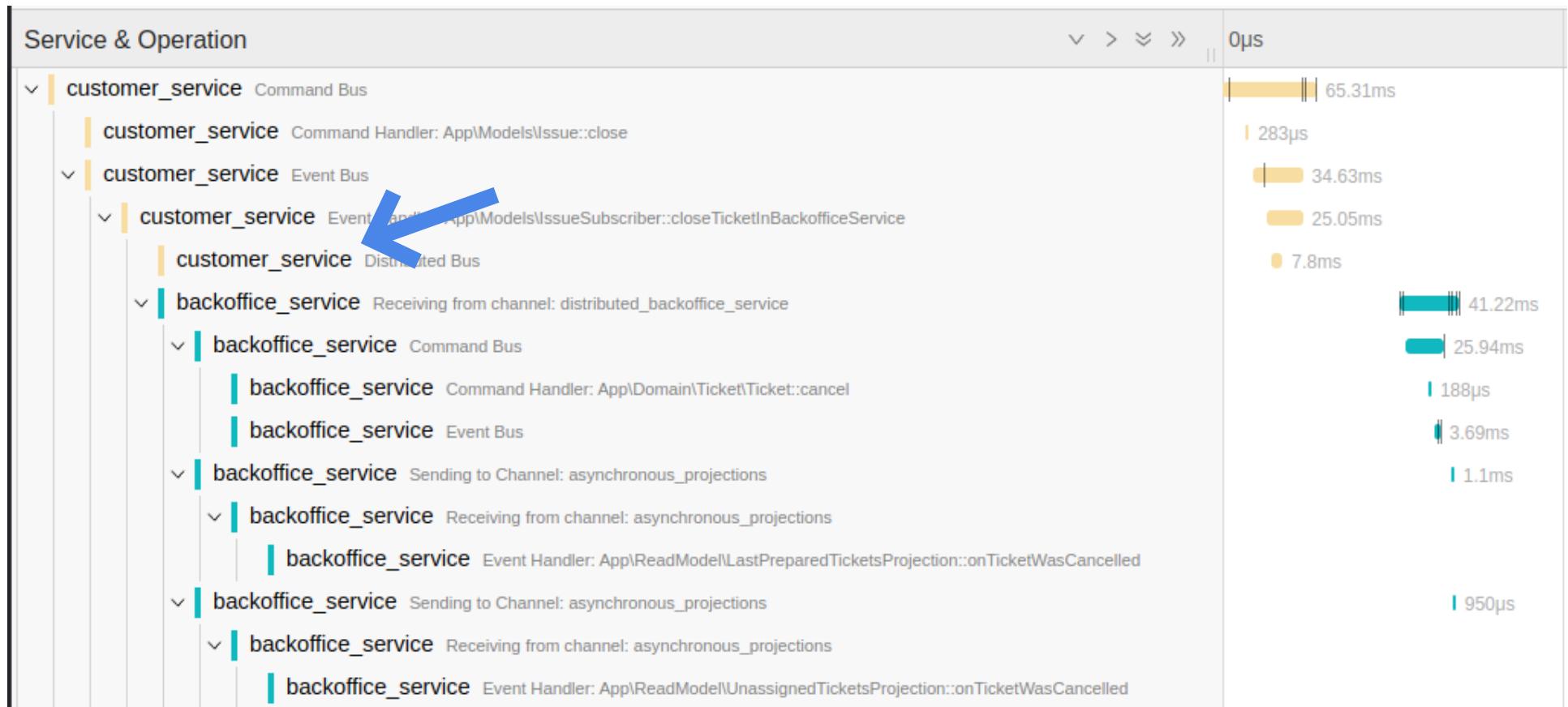
One more important ingredient: Tracing



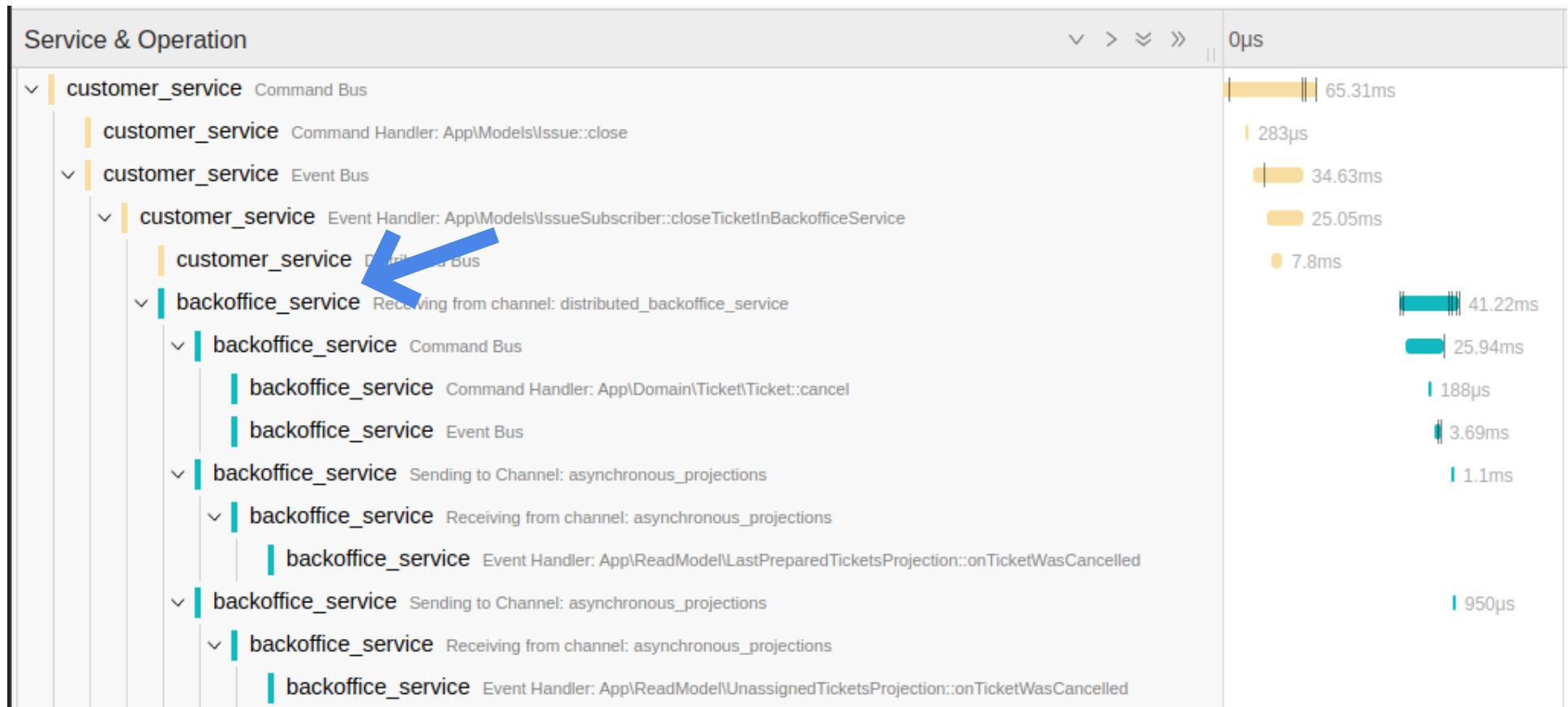
One more important ingredient: Tracing



One more important ingredient: Tracing

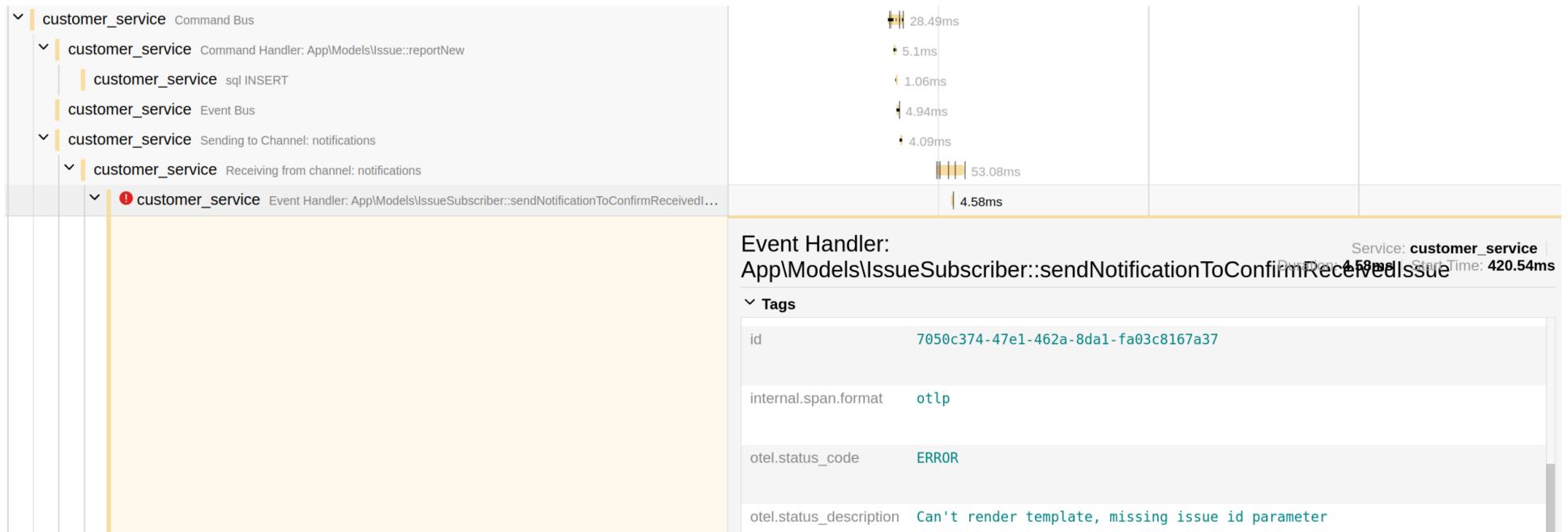


One more important ingredient: Tracing

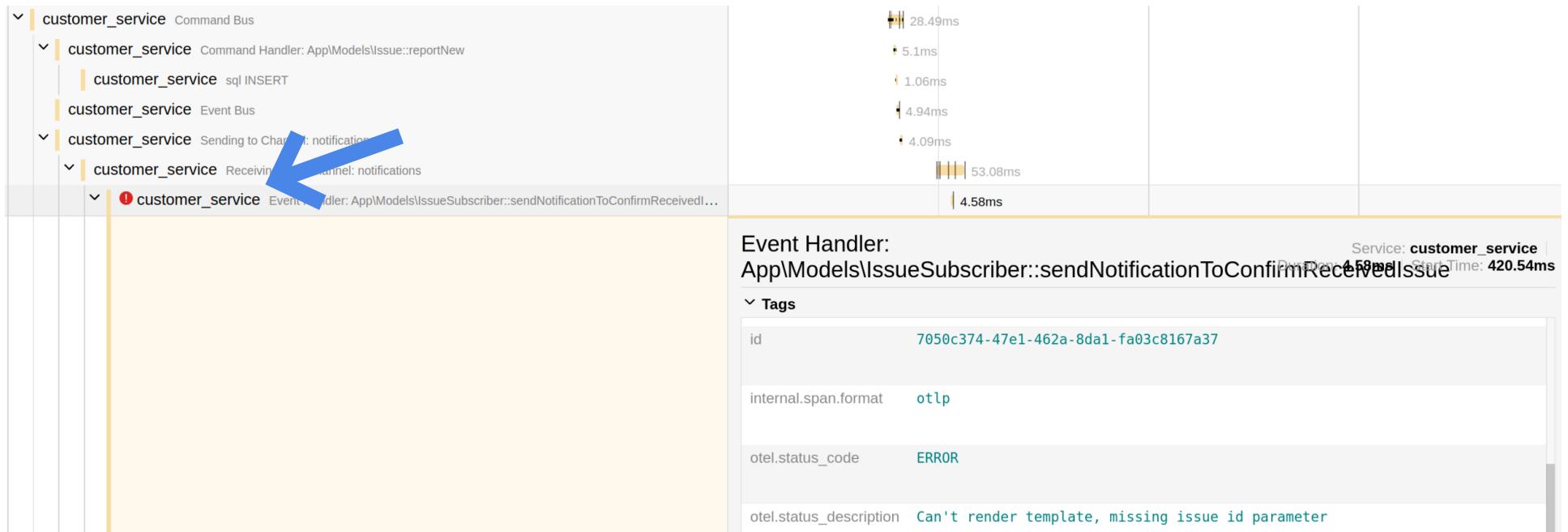


When failure happens recover fast

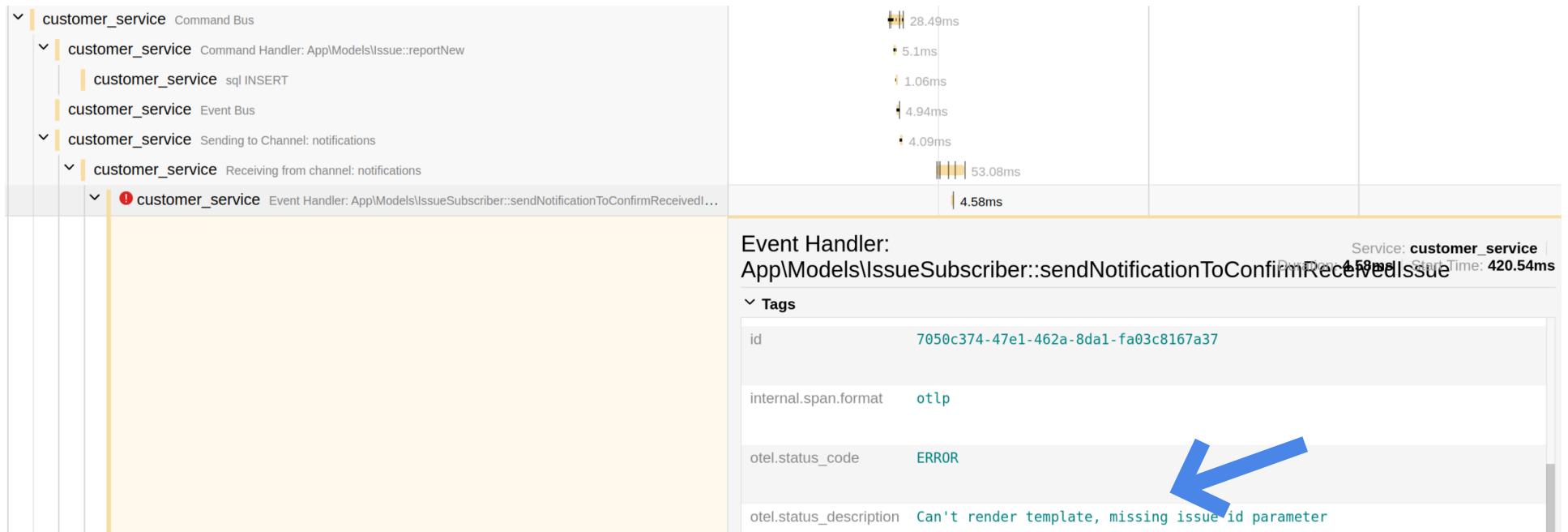
When failure happens recover fast



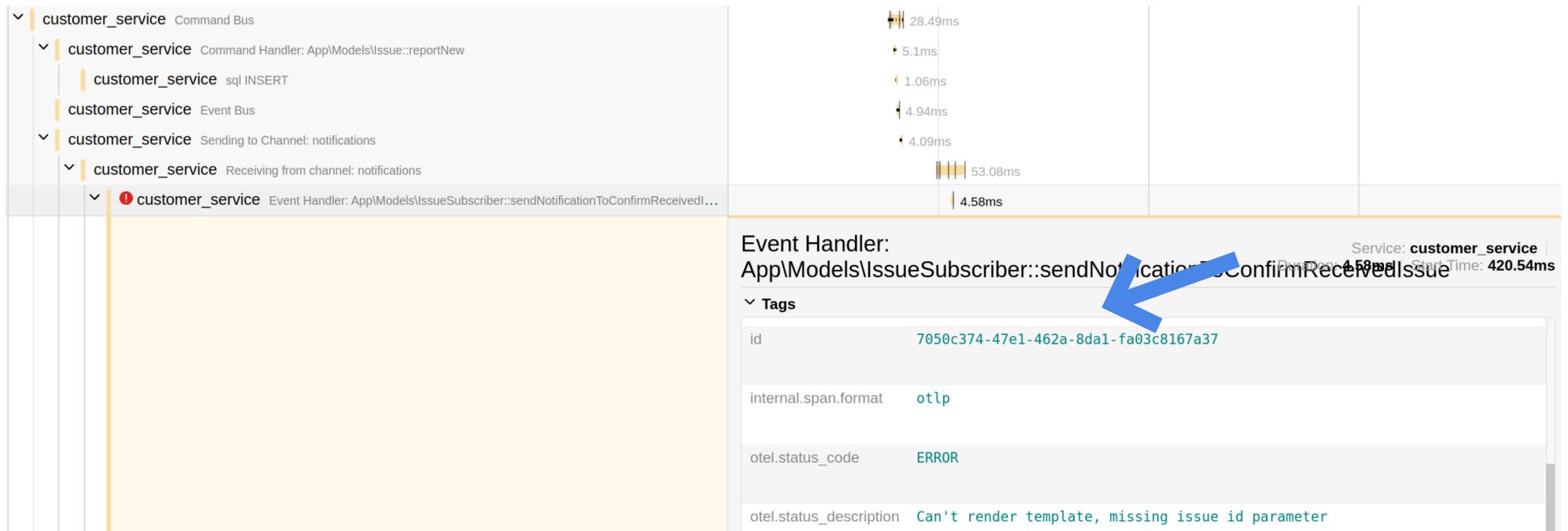
When failure happens recover fast



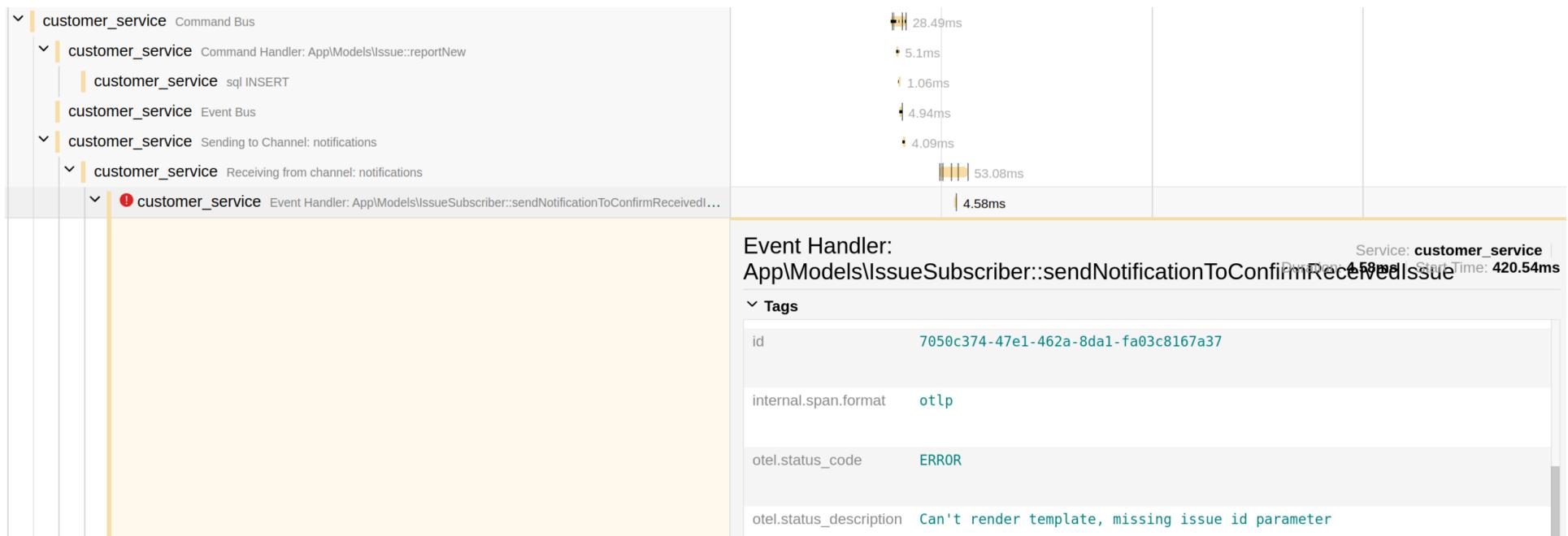
When failure happens recover fast



When failure happens recover fast



When failure happens recover fast



Eccone Pulse

Dashboard

Your Services

customer_service

backoffice_service

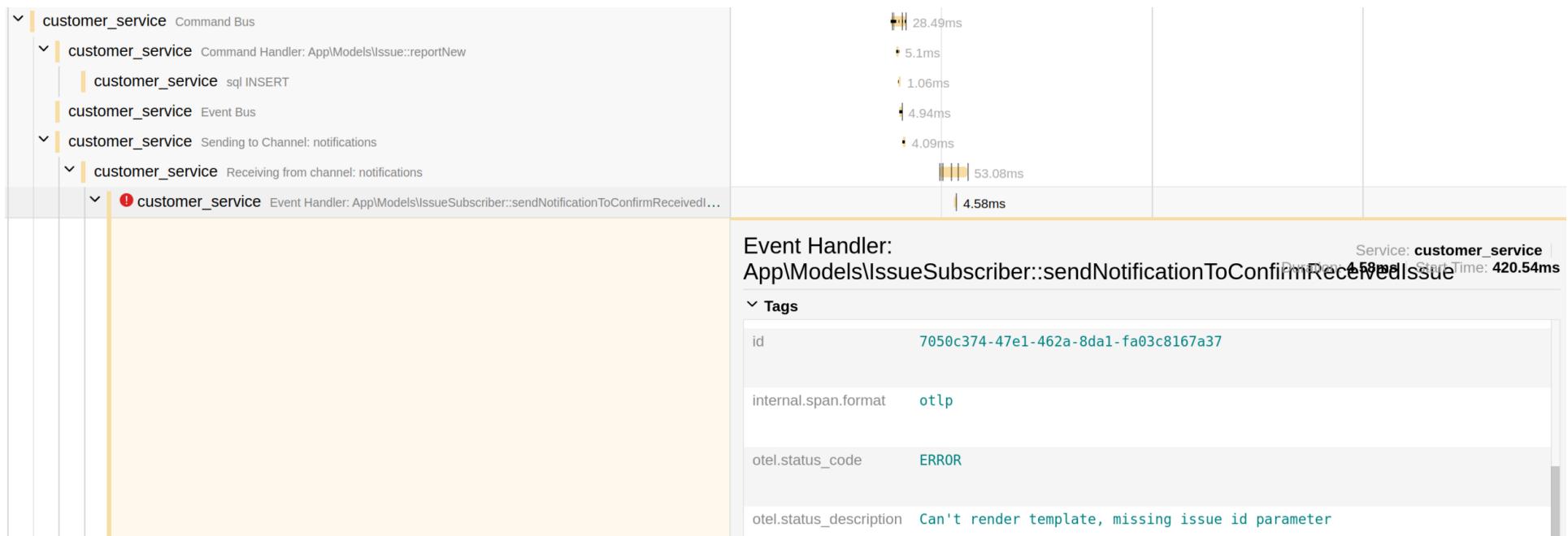
customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
7050c374-47e1-462a-8da1-fa03c8167a37	Can't render template, missing issue id parameter	Show	11-11-2023 15:04:20.000000	<button>Replay</button> <button>Delete</button>

When failure happens recover fast



Ectone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id

Error Message

7050c374-47e1-462a-8da1-fa03c8167a37

Can't render template, missing issue id parameter

Stacktrace

Occurred At

Actions

Show

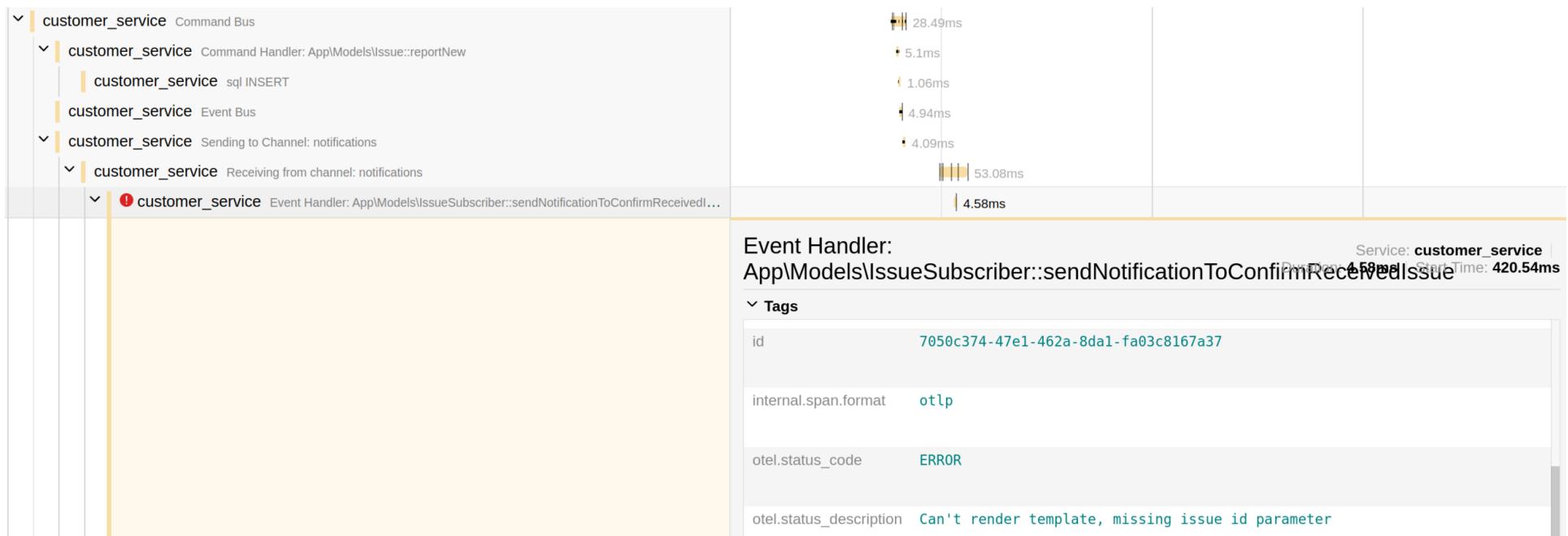
11-11-2023 15:04:20.000000

Replay

Delete



When failure happens recover fast



Ectone Pulse

Dashboard

Your Services

customer_service

backoffice_service

customer_service - Errors

Replay all messages

1

Message Id	Error Message	Stacktrace	Occurred At	Actions
7050c374-47e1-462a-8da1-fa03c8167a37	Can't render template, missing issue id parameter	Show	11-11-2023 15:04:20.000000	<button>Replay</button> <button>Delete</button>

When failure happens recover fast



Resilient Messaging Questions and Workshop (Open in separate IDE)



What happens under the hood?



Configuration & Bootstrap code



Integration & Boilerplate code



Resiliency & recoverability code

What happens under the hood?

- Connecting Event Handlers
 - Queue creation
 - Queue routings
-



Integration & Boilerplate code



Resiliency & recoverability code

What happens under the hood?



- Connecting Event Handlers
 - Queue creation
 - Queue routings
-



- Sending and Consuming from RabbitMQ
 - Message Payload Conversion
 - Message deduplication
-



Resiliency & recoverability code

What happens under the hood?



- Connecting Event Handlers
 - Queue creation
 - Queue routings
-

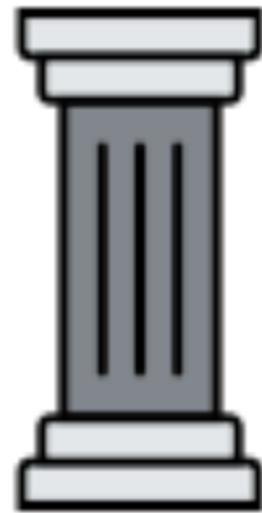


- Sending and Consuming from RabbitMQ
 - Message Payload Conversion
 - Message deduplication
-



- Sending and Consumption retries
 - Message Isolation
 - DLQ and message replays

Building Blocks



**Prepare the
exercise**

Building Blocks in Ecotone

Building Blocks in Ecotone



Command/Event/Query
Handlers

Building Blocks in Ecotone



Command/Event/Query
Handlers



Aggregates / Sagas

Building Blocks in Ecotone



Command/Event/Query
Handlers



Aggregates / Sagas



Event Sourcing
Projections

Much of technical code

```
final class OrderService
{
    public function __construct(
        private OrderRepository $orderRepository, private ProductRepository $productRepository,
        private Clock $clock, private EventBus $eventBus
    ) {}

    #[CommandHandler]
    public function placeOrder(PlaceOrder $command): void
    {
        $order = Order::create(
            $command->userId, $command->shippingAddress,
            $this->getProductDetails($command), $this->clock
        );
        /** Storing order in database */
        $this->orderRepository->save($order);

        /** Publish event indicating that Order Was Placed */
        $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
    }
}
```

Much of technical code

```
final class OrderService
{
    public function __construct(
        private OrderRepository $orderRepository, private ProductRepository $productRepository,
        private Clock $clock, private EventBus $eventBus
    ) {}

#[CommandHandler]
public function placeOrder(PlaceOrder $command): void
{
    $order = Order::create(
        $command->userId, $command->shippingAddress,
        $this->getProductDetails($command), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Publish event indicating that Order Was Placed */
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
}
```

Much of technical code

```
final class OrderService
{
    public function __construct(
        private OrderRepository $orderRepository, private ProductRepository $productRepository,
        private Clock $clock, private EventBus $eventBus
    ) {}

#[CommandHandler]
public function placeOrder(PlaceOrder $command): void
{
    $order = Order::create(
        $command->userId, $command->shippingAddress,
        $this->getProductDetails($command), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Publish event indicating that Order Was Placed */
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
}
```

Much of technical code

```
final class OrderService
{
    public function __construct(
        private OrderRepository $orderRepository, private ProductRepository $productRepository,
        private Clock $clock, private EventBus $eventBus
    ) {}

#[CommandHandler]
public function placeOrder(PlaceOrder $command): void
{
    $order = Order::create(
        $command->userId, $command->shippingAddress,
        $this->getProductDetails($command), $this->clock
    );
    /** Storing order in database */
    $this->orderRepository->save($order);

    /** Publish event indicating that Order Was Placed */
    $this->eventBus->publish(new OrderWasPlaced($order->getOrderId()));
}
```

Orchestration



Introduce Aggregate

```
#[Aggregate]
final class Order
{
    use WithAggregateEvents;

    private function __construct(
        #[AggregateIdentifier] private \UuidInterface $orderId,
        private \UuidInterface $userId,
        private \DateTimeImmutable $orderAt)
    {
        $this->recordThat(new OrderWasPlaced($this->orderId));
    }

#[CommandHandler]
public static function create(PlaceOrder $command, Clock $clock): self
{
    return new self($command->orderId, $command->userId, $clock->getCurrentTime());
}
```

Introduce Aggregate

```
#[Aggregate] ←
final class Order
{
    use WithAggregateEvents;

    private function __construct(
        #[AggregateIdentifier] private \UuidInterface $orderId,
        private \UuidInterface $userId,
        private \DateTimeImmutable $orderAt)
    {
        $this->recordThat(new OrderWasPlaced($this->orderId));
    }

#[CommandHandler]
public static function create(PlaceOrder $command, Clock $clock): self
{
    return new self($command->orderId, $command->userId, $clock->getCurrentTime());
}
```

Introduce Aggregate

```
#[Aggregate]
final class Order
{
    use WithAggregateEvents;

    private function __construct(
        #[AggregateIdentifier] private UuidInterface $orderId,
        private UuidInterface $userId,
        private \DateTimeImmutable $orderAt)
    {
        $this->recordThat(new OrderWasPlaced($this->orderId));
    }

#[CommandHandler]
public static function create(PlaceOrder $command, Clock $clock): self
{
    return new self($command->orderId, $command->userId, $clock->getCurrentTime());
}
```

Introduce Aggregate

```
#[Aggregate]
final class Order
{
    use WithAggregateEvents;

    private function __construct(
        #[AggregateIdentifier] private UuidInterface $orderId,
        private UuidInterface $userId,
        private \DateTimeImmutable $orderAt)
    {
        $this->recordThat(new OrderWasPlaced($this->orderId));
    }

    #[CommandHandler]
    public static function create(PlaceOrder $command, Clock $clock): self
    {
        return new self($command->orderId, $command->userId, $clock->getCurrentTime());
    }
}
```

Introduce Aggregate

```
#[Aggregate]
final class Order
{
    use WithAggregateEvents;

    private function __construct(
        #[AggregateIdentifier] private UuidInterface $orderId,
        private UuidInterface $userId,
        private \DateTimeImmutable $orderAt)
    {
        $this->recordThat(new OrderWasPlaced($this->orderId));
    }

#[CommandHandler]
public static function create(PlaceOrder $command, Clock $clock): self
{
    return new self($command->orderId, $command->userId, $clock->getCurrentTime());
}
```



Repository

```
#[Repository]
final class InMemoryOrderRepository implements StandardRepository
{
    /** @var Order[] */
    private array $orders;

    public function findBy(string $aggregateClassName, array $identifiers): ?object
    {
        return $this->orders[$identifiers['orderId']] ?? null;
    }

    public function save(
        array $identifiers, object $aggregate,
        array $metadata, ?int $versionBeforeHandling
    ): void
    {
        $this->orders[$identifiers['orderId']] = $aggregate;
    }

    public function canHandle(string $aggregateClassName): bool
    {
        return $aggregateClassName === Order::class;
    }
}
```

Repository

```
#[Repository] ←
final class InMemoryOrderRepository implements StandardRepository
{
    /** @var Order[] */
    private array $orders;

    public function findBy(string $aggregateClassName, array $identifiers): ?object
    {
        return $this->orders[$identifiers['orderId']] ?? null;
    }

    public function save(
        array $identifiers, object $aggregate,
        array $metadata, ?int $versionBeforeHandling
    ): void
    {
        $this->orders[$identifiers['orderId']] = $aggregate;
    }

    public function canHandle(string $aggregateClassName): bool
    {
        return $aggregateClassName === Order::class;
    }
}
```

Repository

```
#[Repository]
final class InMemoryOrderRepository implements StandardRepository
{
    /** @var Order[] */
    private array $orders;

    public function findBy(string $aggregateClassName, array $identifiers): ?object
    {
        return $this->orders[$identifiers['orderId']] ?? null;
    }

    public function save(
        array $identifiers, object $aggregate,
        array $metadata, ?int $versionBeforeHandling
    ): void
    {
        $this->orders[$identifiers['orderId']] = $aggregate;
    }

    public function canHandle(string $aggregateClassName): bool
    {
        return $aggregateClassName === Order::class;
    }
}
```

Repository

```
#[Repository]
final class InMemoryOrderRepository implements StandardRepository
{
    /** @var Order[] */
    private array $orders;

    public function findBy(string $aggregateClassName, array $identifiers): ?object
    {
        return $this->orders[$identifiers['orderId']] ?? null;
    }

    public function save(
        array $identifiers, object $aggregate,
        array $metadata, ?int $versionBeforeHandling
    ): void
    {
        $this->orders[$identifiers['orderId']] = $aggregate;
    }

    public function canHandle(string $aggregateClassName): bool
    {
        return $aggregateClassName === Order::class;
    }
}
```

Repository

```
#[Repository]
final class InMemoryOrderRepository implements StandardRepository
{
    /** @var Order[] */
    private array $orders;

    public function findBy(string $aggregateClassName, array $identifiers): ?object
    {
        return $this->orders[$identifiers['orderId']] ?? null;
    }

    public function save(←
        array $identifiers, object $aggregate,
        array $metadata, ?int $versionBeforeHandling
    ): void
    {
        $this->orders[$identifiers['orderId']] = $aggregate;
    }

    public function canHandle(string $aggregateClassName): bool
    {
        return $aggregateClassName === Order::class;
    }
}
```

Repository

```
#[Repository]
final class InMemoryOrderRepository implements StandardRepository
{
    /** @var Order[] */
    private array $orders;

    public function findBy(string $aggregateClassName, array $identifiers): ?object
    {
        return $this->orders[$identifiers['orderId']] ?? null;
    }

    public function save(
        array $identifiers, object $aggregate,
        array $metadata, ?int $versionBeforeHandling
    ): void
    {
        $this->orders[$identifiers['orderId']] = $aggregate;
    }

    public function canHandle(string $aggregateClassName): bool
    {
        return $aggregateClassName === Order::class;
    }
}
```



API stays the same

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);

    $this->commandBus->send($command);

    return new Response();
}
```

API stays the same

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);

    $this->commandBus->send($command);

    return new Response();
}
```



API stays the same

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);

    $this->commandBus->send($command);

    return new Response();
}
```



API stays the same

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);
    
    $this->commandBus->send($command);

    return new Response();
}
```

Actions on Aggregate

```
final readonly class CancelOrder
{
    public function __construct(
        public UuidInterface $orderId
    ) {}

}
```

Actions on Aggregate

```
final readonly class CancelOrder

{
    public function __construct(
        public UuidInterface $orderId
    ) {}

final class Order
{
    #[AggregateIdentifier] private UuidInterface $orderId;
    private bool $isCancelled = false;

    #[CommandHandler]
    public function cancel(CancelOrder $command): void
    {
        $this->isCancelled = true;
    }
}
```

Actions on Aggregate

```
final readonly class CancelOrder

{
    public function __construct(
        public UuidInterface $orderId
    ) {}

final class Order
{
    #[AggregateIdentifier] private UuidInterface $orderId;
    private bool $isCancelled = false;

    #[CommandHandler]
    public function cancel(CancelOrder $command): void
    {
        $this->isCancelled = true;
    }
}
```

Actions on Aggregate

```
final readonly class CancelOrder

{
    public function __construct(
        public UuidInterface $orderId
    ) {}

final class Order
{
    #[AggregateIdentifier] private UuidInterface $orderId;
    private bool $isCancelled = false;

    #[CommandHandler]
    public function cancel(CancelOrder $command): void
    {
        $this->isCancelled = true;
    }
}
```



Actions on Aggregate

```
final readonly class CancelOrder

{
    public function __construct(
        public UuidInterface $orderId
    ) {}

final class Order
{
    #[AggregateIdentifier] private UuidInterface $orderId;
    private bool $isCancelled = false;

    #[CommandHandler]
    public function cancel(CancelOrder $command): void
    {
        $this->isCancelled = true;
    }
}
```



Routing based Actions

```
# [CommandHandler]
public function cancel(CancelOrder $command): void
{
    $this->isCancelled = true;
}
```

Routing based Actions

```
# [CommandHandler]
public function cancel(CancelOrder $command): void
{
    $this->isCancelled = true;
}
```



Routing based Actions

```
# [CommandHandler("order.cancel")]
public function cancel(): void
{
    $this->isCancelled = true;
}
```

Routing based Actions

```
# [CommandHandler("order.cancel")]
public function cancel(): void
{
    $this->isCancelled = true;
}
```

Routing based Actions

```
# [CommandHandler("order.cancel")]
public function cancel(): void
{
    $this->isCancelled = true;
}
```

Routing based Actions

```
# [CommandHandler("order.cancel")]
public function cancel(): void
{
    $this->isCancelled = true;
}

$this->commandBus->sendWithRouting(
    routingKey: "order.cancel",
    metadata: ["aggregate.id" => $orderId]
);
```

Routing based Actions

```
# [CommandHandler("order.cancel")]
public function cancel(): void
{
    $this->isCancelled = true;
}

$this->commandBus->sendWithRouting(
    routingKey: "order.cancel",
    metadata: ["aggregate.id" => $orderId]
);
```

Routing based Actions

```
# [CommandHandler("order.cancel")]
public function cancel(): void
{
    $this->isCancelled = true;
}

$this->commandBus->sendWithRouting(
    routingKey: "order.cancel",
    metadata: ["aggregate.id" => $orderId]
);
```

Aggregates subscribing to Events

```
# [EventHandler]
public function decreaseStock(
    OrderWasPlaced $event, ProductStockRepository $productStockRepository
): void
{
    $productStock = $productStockRepository->get($event->productId);
    $productStock->decreaseStock();
    $productStockRepository->save($productStock);
}
```

Aggregates subscribing to Events

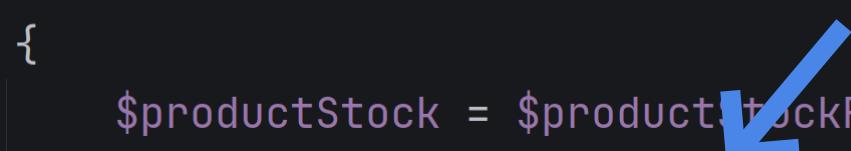
```
# [EventHandler]   
public function decreaseStock(  
    OrderWasPlaced $event, ProductStockRepository $productStockRepository  
): void  
{  
    $productStock = $productStockRepository->get($event->productId);  
    $productStock->decreaseStock();  
    $productStockRepository->save($productStock);  
}
```

Aggregates subscribing to Events

```
# [EventHandler]
public function decreaseStock(
    OrderWasPlaced $event, ProductStockRepository $productStockRepository
): void
{
    $productStock = $productStockRepository->get($event->productId);
    $productStock->decreaseStock();
    $productStockRepository->save($productStock);
}
```



Aggregates subscribing to Events

```
# [EventHandler]
public function decreaseStock(
    OrderWasPlaced $event, ProductStockRepository $productStockRepository
): void
{
    $productStock = $productStockRepository->get($event->productId);

    $productStock->decreaseStock();
    $productStockRepository->save($productStock);
}
```

Aggregates subscribing to Events

```
# [EventHandler]
public function decreaseStock(
    OrderWasPlaced $event, ProductStockRepository $productStockRepository
): void
{
    $productStock = $productStockRepository->get($event->productId);
    $productStock->decreaseStock();
    $productStockRepository->save($productStock);
}
```

Aggregates subscribing to Events

```
# [EventHandler]
public function decreaseStock(
    OrderWasPlaced $event, ProductStockRepository $productStockRepository
): void
{
    $productStock = $productStockRepository->get($event->productId);
    $productStock->decreaseStock();
    $productStockRepository->save($productStock);
}
```

Orchestration



Get rid of orchestration

```
final readonly class OrderWasPlaced
{
    public function __construct(
        public string $orderId,
        public string $productId
    ) {}

}
```

Get rid of orchestration

```
final readonly class OrderWasPlaced
{
    public function __construct(
        public string $orderId,
        public string $productId
    ) {}

}
```

Get rid of orchestration

```
public function __construct(  
    public string $orderId,  
    public string $productId  
) {}
```

Get rid of orchestration

```
public function __construct(  
    public string $orderId,  
    public string $productId  
) {}  
  
#[Aggregate]  
final class ProductStock  
{  
    public function __construct(  
        #[Identifier] private string $productId,  
        private int $productStockCount  
    ) {}  
  
    #[EventHandler]  
    public function decreaseStock(OrderWasPlaced $event): void  
    {  
        $this->productStockCount--;  
    }  
}
```

Get rid of orchestration

```
public function __construct(  
    public string $orderId,  
    public string $productId  
) {}  
  
#[Aggregate]  
final class ProductStock  
{  
    public function __construct(  
        #[Identifier] private string $productId,  
        private int $productStockCount  
    ) {}  
  
    #[EventHandler]  
    public function decreaseStock(OrderWasPlaced $event): void  
    {  
        $this->productStockCount--;  
    }  
}
```



Get rid of orchestration

```
public function __construct(  
    public string $orderId,  
    public string $productId  
) {}  
  
#[Aggregate]  
final class ProductStock  
{  
    public function __construct(  
        #[Identifier] private string $productId,  
        private int $productStockCount  
    ) {}  
  
    #[EventHandler]  
    public function decreaseStock(OrderWasPlaced $event): void  
    {  
        $this->productStockCount--;  
    }  
  
}
```



Get rid of orchestration

```
public function __construct(  
    public string $orderId,  
    public string $productId  
) {}  
  
#[Aggregate]  
final class ProductStock  
{  
    public function __construct(  
        #[Identifier] private string $productId,  
        private int $productStockCount  
    ) {}  
  
    #[EventHandler]  
    public function decreaseStock(OrderWasPlaced $event): void  
    {  
        $this->productStockCount--;  
    }  
}
```

Get rid of orchestration

```
public function __construct(  
    public string $orderId,  
    public string $productId  
) {}  
  
#[Aggregate]  
final class ProductStock  
{  
    public function __construct(  
        #[Identifier] private string $productId,  
        private int $productStockCount  
    ) {}  
  
    #[EventHandler]  
    public function decreaseStock(OrderWasPlaced $event): void  
    {  
        $this->productStockCount--;  
    }  
}
```



Routing based Events

```
# [NamedEvent('order.placed')]
final readonly class OrderWasPlaced
{
    public function __construct(
        public string $orderId,
        public string $productId
    ) {}

}
```

Routing based Events



```
#[NamedEvent('order.placed')]  
final readonly class OrderWasPlaced  
{  
    public function __construct(  
        public string $orderId,  
        public string $productId  
    ) {}  
}
```

Routing based Events

```
#[Aggregate]
final class ProductStock
{
    public function __construct(
        #[Identifier] private string $productId,
        private int $productStockCount
    ) {
    }

#[EventHandler(listenTo: "order.placed")]
public function decreaseStock(): void
{
    $this->productStockCount--;
}
```

Routing based Events

```
#[Aggregate]
final class ProductStock
{
    public function __construct(
        #[Identifier] private string $productId,
        private int $productStockCount
    ) {
    }

#[EventHandler(listenTo: "order.placed")]
public function decreaseStock(): void
{
    $this->productStockCount--;
}
```



Routing based Events

```
#[Aggregate]
final class ProductStock
{
    public function __construct(
        #[Identifier] private string $productId,
        private int $productStockCount
    ) {
    }

#[EventHandler(listenTo: "order.placed")]
public function decreaseStock(): void
{
    $this->productStockCount--;
}
```



Routing based Events

```
#[Aggregate]
final class ProductStock
{
    public function __construct(
        #[Identifier] private string $productId,
        private int $productStockCount
    ) {
    }

#[EventHandler(listenTo: "order.placed")]
public function decreaseStock(): void
{
    $this->productStockCount--;
}
```



Avoiding Manual Transformations

```
#[CommandHandler("order.place")]
public static function create(
    PlaceOrder $command
): self
{
    return new self(
        $command->orderId,
        $command->productName
    );
}
```

Avoiding Manual Transformations

```
#[CommandHandler("order.place")]
public static function create(
    PlaceOrder $command
): self
{
    return new self(
        $command->orderId,
        $command->productName
    );
}
```

Avoiding Manual Transformations

```
#[CommandHandler("order.place")]
public static function create(
    PlaceOrder $command
): self
{
    return new self(
        $command->orderId,
        $command->productName
    );
}
```



Avoiding Manual Transformations

```
#[CommandHandler("order.place")]
public static function create(
    PlaceOrder $command
): self
{
    return new self(
        $command->orderId, ←
        $command->productName
    );
}
```

Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);

    $this->commandBus->send($command);

    return new Response();
}
```

Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);

    $this->commandBus->send($command);

    return new Response();
}
```



Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);

    $this->commandBus->send($command);

    return new Response();
}
```



Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $data = \json_decode($request->getContent(), associative: true);
    $command = $this->createCommandFrom($data);
    $this->commandBus->send($command);
    return new Response();
}
```



Associative arrays are great for mapping JSON data to objects.

But what if you need to map objects to JSON?

That's where `Response::json()` comes in.

Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $this->commandBus->sendWithRouting(
        routingKey: 'order.place',
        $request->getContent(),
        commandMediaType: 'application/json'
    );
    return new Response();
}
```

Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $this->commandBus->sendWithRouting(
        routingKey: 'order.place',
        $request->getContent(),
        commandMediaType: 'application/json'
    );
    return new Response();
}
```

Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $this->commandBus->sendWithRouting(
        routingKey: 'order.place',
        $request->getContent(), 
        commandMediaType: 'application/json'
    );
    return new Response();
}
```

Avoiding Manual Transformations

```
public function placeOrder(Request $request): Response
{
    $this->commandBus->sendWithRouting(
        routingKey: 'order.place',
        $request->getContent(),
        commandMediaType: 'application/json' 
```

With bad architecture, don't expect Domain focus



With good architecture,
Domain comes forward naturally



Building Blocks Questions and Workshop (Open in separate IDE)



What we achieved?



Configuration & Bootstrap code



Integration & Boilerplate code



Resiliency & recoverability code

What we achieved?



- Aggregates can be connect
to Messaging directly in a POPO way



Integration & Boilerplate code



Resiliency & recoverability code

What we achieved?



- Aggregates can be connect to Messaging directly in a POPO way



- Application/Service layer can be removed
- Events can be recorded in Aggregate
- Building Blocks can be connected seamlessly



Resiliency & recoverability code

What we achieved?



- Aggregates can be connect to Messaging directly in a POPO way

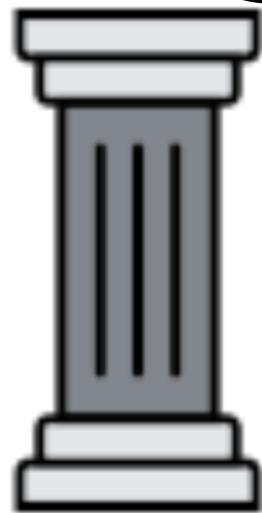


- Application/Service layer can be removed
- Events can be recorded in Aggregate
- Building Blocks can be connected seamlessly



- Is done by used messaging

Testing Messaging



Prepare the
exercise

Rules of maintainable tests

Rules of maintainable tests



Isolate tested flow

Rules of maintainable tests



Isolate tested flow



Test business use case

Rules of maintainable tests



Isolate tested flow



Test business use case



Keep tests close to production
like run

1. Isolate tested flows

Bootstrap Ecotone Lite

Bootstrap Ecotone Lite

```
# [CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```

Bootstrap Ecotone Lite

```
# [CommandHandler]  
public function placeOrder(  
    PlaceOrder $placeOrder,  
    OrderRepository $orderRepository): void  
{  
    $order = Order::create($placeOrder->productName);  
    $orderRepository->save($order);  
}
```



Bootstrap Ecotone Lite

```
# [CommandHandler]  
public function placeOrder(  
    PlaceOrder $placeOrder,  
    OrderRepository $orderRepository): void  
{  
    $order = Order::create($placeOrder->productName);  
    $orderRepository->save($order);  
}
```



Bootstrap Ecotone Lite

```
# [CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```

```
public function test_placing_an_order(): void
{
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [OrderService::class],
        [new OrderService(), new InMemoryOrderRepository()]
    );
}
```

Bootstrap Ecotone Lite

```
# [CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```

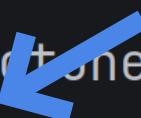
```
public function test_placing_an_order(): void
{
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [OrderService::class],
        [new OrderService(), new InMemoryOrderRepository()]
);
```



Bootstrap Ecotone Lite

```
# [CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```

```
public function test_placing_an_order(): void
{
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [OrderService::class],
        [new OrderService(), new InMemoryOrderRepository()]
);
```



Bootstrap Ecotone Lite

```
# [CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```

```
public function test_placing_an_order(): void
{
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [OrderService::class],           ←
        [new OrderService(), new InMemoryOrderRepository()])
};
```

Executing Test Scenario

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService(), $inMemoryRepository]  
);  
  
$ecotoneLite->sendCommand(  
    new PlaceOrder($orderId, productName: 'milk')  
);  
  
$this->assertNotNull(  
    $inMemoryRepository->get($orderId)  
);
```

Executing Test Scenario

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService(), $inMemoryRepository]  
);  
  
$ecotoneLite->sendCommand(  
    new PlaceOrder($orderId, productName: 'milk')  
);  
  
$this->assertNotNull(  
    $inMemoryRepository->get($orderId)  
);
```

Executing Test Scenario

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService(), $inMemoryRepository]  
);  
  
$ecotoneLite->sendCommand(  
    new PlaceOrder($orderId, productName: 'milk')  
);  
  
$this->assertNotNull(  
    $inMemoryRepository->get($orderId)  
);
```



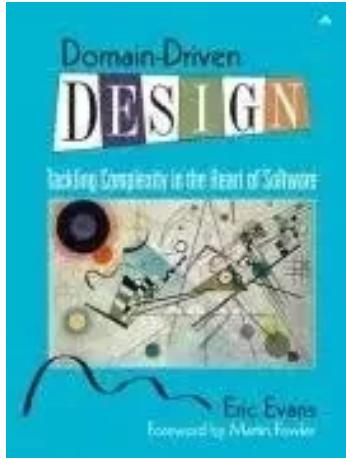
Executing Test Scenario

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService(), $inMemoryRepository]  
);  
  
$ecotoneLite->sendCommand(  
    new PlaceOrder($orderId, productName: 'milk')  
);  
  
$this->assertNotNull(  
    $inMemoryRepository->get($orderId)  
);
```

2. Test business use case

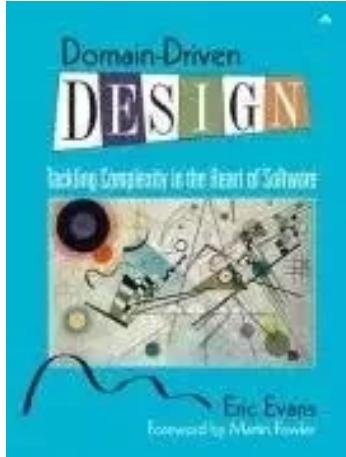
All starts at API level

All starts at API level

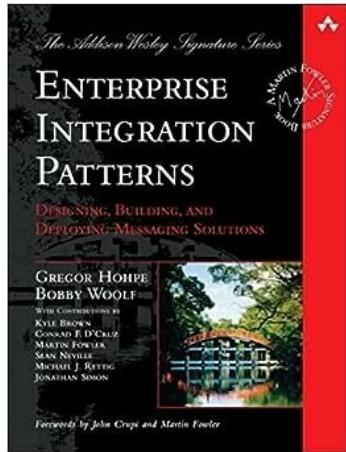


- Aggregates are our API

All starts at API level



- Aggregates are our API



- Messages are our API

Testing Messaging Aggregates

```
#[Aggregate]
final class Order
{
    #[CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
        );
    }
}
```

Testing Messaging Aggregates

```
# [Aggregate] ←
final class Order
{
    # [CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
        );
    }
}
```

Testing Messaging Aggregates

```
# [Aggregate]
final class Order
{
    #[CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
        );
    }
}
```



Testing Messaging Aggregates

```
# [Aggregate]
final class Order
{
    #[CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
        );
    }

    public function test_placing_an_order(): void
    {
        $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
            [Order::class],
            []
        );
    }
}
```

Testing Messaging Aggregates

```
# [Aggregate]
final class Order
{
    #[CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
    }
}

public function test_placing_an_order(): void
{
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [Order::class],
        []
    );
}
```



Testing Messaging Aggregates

```
# [Aggregate]
final class Order
{
    #[CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
    }
}

public function test_placing_an_order(): void
{
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [Order::class],
        []
    );
}
```

Testing Messaging Aggregates

```
# [Aggregate]
final class Order
{
    #[CommandHandler]
    public static function create(PlaceOrder $command): self
    {
        return new self(
            $command->orderId,
            $command->productName
    }

    public function test_placing_an_order(): void
    {
        $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
            [Order::class],
            []
        );
    }
}
```

Testing Aggregate API

```
public function test_placing_an_order(): void
{
    $orderId = Uuid::uuid4()->toString();
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [Order::class]
    );

    $ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));

    $this->assertNotNull(
        $ecotoneLite->getAggregate(className: Order::class, $orderId)
    );
}
```

Testing Aggregate API

```
public function test_placing_an_order(): void
{
    $orderId = Uuid::uuid4()->toString();
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [Order::class]
    );

    $ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));

    $this->assertNotNull(
        $ecotoneLite->getAggregate(className: Order::class, $orderId)
    );
}
```



Testing Aggregate API

```
public function test_placing_an_order(): void
{
    $orderId = Uuid::uuid4()->toString();
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [Order::class]
    );
    $ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));

    $this->assertNotNull(
        $ecotoneLite->getAggregate(className: Order::class, $orderId)
    );
}
```



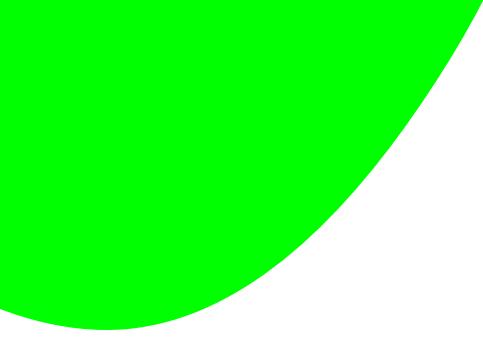
Testing Aggregate API

```
public function test_placing_an_order(): void
{
    $orderId = Uuid::uuid4()->toString();
    $ecotoneLite = EcotoneLite::bootstrapFlowTesting(
        [Order::class]
    );

    $ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));

    $this->assertNotNull(
        $ecotoneLite->getAggregate(className: Order::class, $orderId)
    );
}
```





*3. Keep tests close to
production run*

Synchronous Actions

```
# [Route('/order', 'POST')]  
public function placeOrder(Request $request): Response  
{  
    $this->commandBus->send(new PlaceOrder(  
        $request->get('orderId'),  
        $request->get('productName')  
    ));  
  
    return new Response();  
}
```

Synchronous Actions

```
#Route('/order', 'POST')
public function placeOrder(Request $request): Response
{
    $this->commandBus->send(new PlaceOrder(
        $request->get('orderId'),
        $request->get('productName')
    ));

    return new Response();
}
```



Synchronous Actions

```
# [Route('/order/{orderId}', 'GET')]  
public function getOrder(Request $request): Response  
{  
    $order = $this->queryBus->send(new GetOrderById(  
        $request->get('orderId')  
    ));  
  
    return new Response($order);  
}
```

Synchronous Actions

```
#Route('/order/{orderId}', 'GET')
public function getOrder(Request $request): Response
{
    $order = $this->queryBus->send(new GetOrderById(
        $request->get('orderId')
    ));

    return new Response($order);
}
```



Synchronous Actions

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService, $inMemoryRepository]  
);  
  
$this->assertNotNull(  
    $ecotoneLite  
        ->sendCommand(new PlaceOrder($orderId, productName: 'milk'))  
        ->sendQuery(new GetOrderById($orderId))  
);
```

Synchronous Actions

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService, $inMemoryRepository]  
);  
  
$this->assertNotNull(  
    $ecotoneLite  
        ->sendCommand(new PlaceOrder($orderId, productName: 'milk'))  
        ->sendQuery(new GetOrderById($orderId))  
);
```



Synchronous Actions

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService, $inMemoryRepository]  
);  
  
$this->assertNotNull(  
    $ecotoneLite  
        ->sendCommand(new PlaceOrder($orderId, productName: 'milk'))  
        ->sendQuery(new GetOrderById($orderId))  
);
```

Synchronous Actions

```
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [OrderService::class],  
    [new OrderService, $inMemoryRepository]  
);  
  
$this->assertNotNull(  
    $ecotoneLite  
        ->sendCommand(new PlaceOrder($orderId, productName: 'milk'))  
        ->sendQuery(new GetOrderById($orderId))  
);
```



Asynchronous Actions

```
# [Asynchronous("notifications")]
#[EventHandler(endpointId: 'orderWasPlacedSms')]
public function when(
    OrderWasPlaced $event,
    SmsService $smsService
): void
{
    $smsService->send(
        "Order with id {$event->orderId} was placed."
    );
}
```

Asynchronous Actions

```
# [Asynchronous("notifications")]
#[EventHandler(endpointId: 'orderWasPlacedSms')]
public function when(
    OrderWasPlaced $event,
    SmsService $smsService
): void
{
    $smsService->send(
        "Order with id {$event->orderId} was placed."
    );
}
```

Asynchronous Actions

```
# [Asynchronous("notifications")]
#[EventHandler(endpointId: 'orderWasPlacedSms')]
public function when(
    OrderWasPlaced $event,
    SmsService $smsService
): void
{
    $smsService->send(
        "Order with id {$event->orderId} was placed."
    );
}
```



Asynchronous Actions

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService]
);

$ecotoneLite->sendCommand(
    new PlaceOrder($orderId, productName: 'milk')
);

$this->assertSame(
    expected: 1,
    $stubSmsService->numberOfSmsSent()
);
```

Asynchronous Actions

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService]
);

$ecotoneLite->sendCommand(
    new PlaceOrder($orderId, productName: 'milk')
);

$this->assertSame(
    expected: 1,
    $stubSmsService->numberOfSmsSent()
);
```

Asynchronous Actions

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService]
);

$ecotoneLite->sendCommand(
    new PlaceOrder($orderId, productName: 'milk')
);

$this->assertSame(
    expected: 1,
    $stubSmsService->numberOfSmsSent()
);
```

Asynchronous Actions

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService]
);


$ecotoneLite->sendCommand(
    new PlaceOrder($orderId, productName: 'milk')
);

$this->assertSame(
    expected: 1,
    $stubSmsService->numberOfSmsSent()
);
```

Asynchronous Actions

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService]
);

$ecotoneLite->sendCommand(
    new PlaceOrder($orderId, productName: 'milk')
);

$this->assertSame(
    expected: 1,
    $stubSmsService->numberOfSmsSent()
);
```

More Production like using real Async

More Production like using real Async



Sending and Consuming from
Message Channel (Queue)

More Production like using real Async



Sending and Consuming from
Message Channel (Queue)



Message Serialization and
Deserialization

More Production like using real Async



Sending and Consuming from
Message Channel (Queue)



Message Serialization and
Deserialization



Integration with Message Broker

Full Asynchronous Test

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService],
    enableAsynchronousProcessing: [
        SimpleMessageChannelBuilder::createQueueChannel(
            messageChannelName: 'notifications'
        )
    ]
);

$ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));
$ecotoneLite->run(name: 'notifications');

$this->assertSame(expected: 1, $stubSmsService->numberOfSmsSent());
```

Full Asynchronous Test

```
$stubSmsService = new StubSmsService();  
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(  
    [Order::class, NotificationService::class],  
    [new NotificationService(), $stubSmsService],  
    enableAsynchronousProcessing: [  
        SimpleMessageChannelBuilder::createQueueChannel(  
            messageChannelName: 'notifications'  
        )  
    ]  
);  
  
$ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));  
$ecotoneLite->run(name: 'notifications');  
  
$this->assertSame(expected: 1, $stubSmsService->numberOfSmsSent());
```

Full Asynchronous Test

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService],
    enableAsynchronousProcessing: [
        SimpleMessageChannelBuilder::createQueueChannel(
            messageChannelName: 'notifications'
        )
    ]
);

$ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));
$ecotoneLite->run(name: 'notifications');

$this->assertSame(expected: 1, $stubSmsService->numberOfSmsSent());
```

Full Asynchronous Test

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService],
    enableAsynchronousProcessing: [
        SimpleMessageChannelBuilder::createQueueChannel(
            messageChannelName: 'notifications'
        )
    ]
);

$ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));
$ecotoneLite->run(name: 'notifications');

$this->assertSame(expected: 1, $stubSmsService->numberOfSmsSent());
```



Full Asynchronous Test

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService],
    enableAsynchronousProcessing: [
        SimpleMessageChannelBuilder::createQueueChannel(
            messageChannelName: 'notifications'
        )
    ]
);

$ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));
$ecotoneLite->run(name: 'notifications');

$this->assertSame(expected: 1, $stubSmsService->numberOfSmsSent());
```

Full Asynchronous Test

```
$stubSmsService = new StubSmsService();
$ecotoneLite = EcotoneLite::bootstrapFlowTesting(
    [Order::class, NotificationService::class],
    [new NotificationService(), $stubSmsService],
    enableAsynchronousProcessing: [
        SimpleMessageChannelBuilder::createQueueChannel(
            messageChannelName: 'notifications'
        )
    ]
);

$ecotoneLite->sendCommand(new PlaceOrder($orderId, productName: 'milk'));
$ecotoneLite->run(name: 'notifications');

$this->assertSame(expected: 1, $stubSmsService->numberOfSmsSent());
```

Higher Level Abstraction Testing



Testing Messaging Questions and Workshop



What we achieved?



Configuration & Bootstrap code



Integration & Boilerplate code



Resiliency & recoverability code

What we achieved?



– Bootstrapping and configuration happens in Ecotone Lite



Integration & Boilerplate code



Resiliency & recoverability code

What we achieved?



- Bootstrapping and configuration happens in Ecotone Lite
-



- Integration is production like done by Ecotone
 - We reduce boilerplate to minimum as Ecotone provides sensible defaults
-



- Resiliency & recoverability code

What we achieved?



- Bootstrapping and configuration happens in Ecotone Lite
-



- Integration is production like done by Ecotone
 - We reduce boilerplate to minimum as Ecotone provides sensible defaults
-

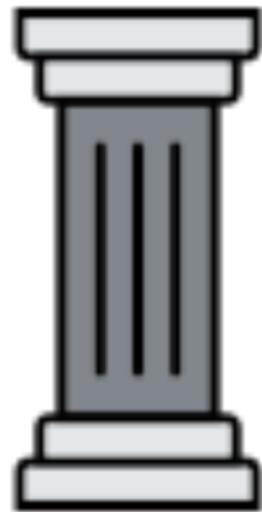


- Is done and tested in Messaging



Thank you

Tips, Tricks and Features



**Start small with your
current code base**

Start small with your current code base



Integrates with
Symfony and Laravel

Start small with your current code base



Integrates with
Symfony and Laravel



Can be used stand-alone or with any
other framework with Ecotone Lite

Start small with your current code base



Integrates with
Symfony and Laravel



Can be used stand-alone or with any
other framework with Ecotone Lite



Can be introduced for
single feature first

Message Headers (Metadata)

```
#Route('/order', 'POST')
public function placeOrder(Request $request): Response
{
    $executorId = $this->authorizationService->getCurrentUserId();
    $this->commandBus->send(new CancelOrder($request->get('orderId')),
    [
        'executorId' => $executorId
    ]);

    return new Response();
}
```

Message Headers (Metadata)

```
#Route('/order', 'POST')
public function placeOrder(Request $request): Response
{
    $executorId = $this->authorizationService->getCurrentUserId();
    $this->commandBus->send(new CancelOrder($request->get('orderId')),
    [
        'executorId' => $executorId
    ]);

    return new Response();
}
```



Message Headers (Metadata)

```
# [CommandHandler]
public function cancel(
    CancelOrder $command,
    #[Header("executorId")] string $executorId
): void
{
    if ($this->ownedBy !== $executorId) {
        throw new \InvalidArgumentException(
            message: "Unauthorized order cancellation."
        );
}
```

Message Headers (Metadata)

```
# [CommandHandler]
public function cancel(
    CancelOrder $command,
    #[Header("executorId")]
    string $executorId
): void
{
    if ($this->ownedBy !== $executorId) {
        throw new \InvalidArgumentException(
            message: "Unauthorized order cancellation."
        );
}
```

Message Consumer Deduplication

Message Consumer Deduplication



Consume Message
from Message Broker

Message Consumer Deduplication



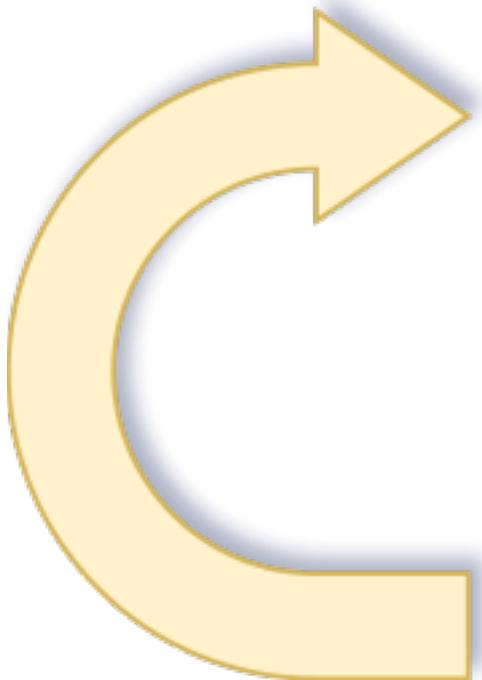
Handle Message

Message Consumer Deduplication

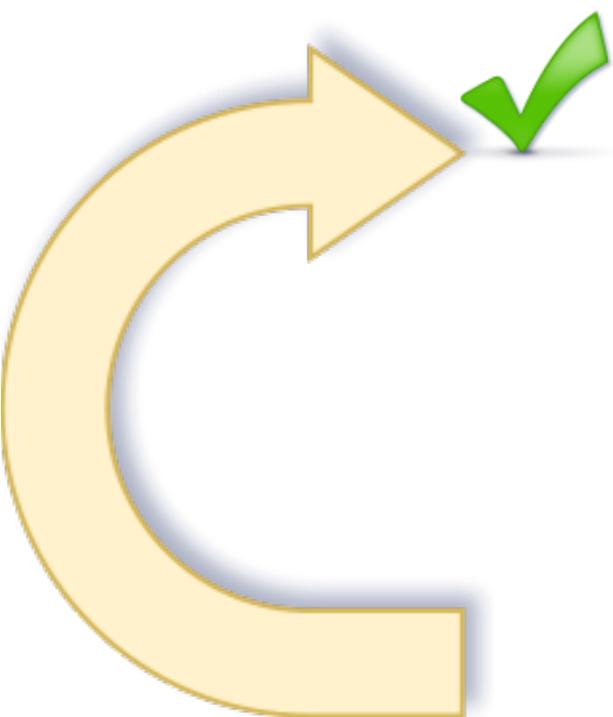


Acknowledge Message
in Message Broker

Message Consumer Deduplication

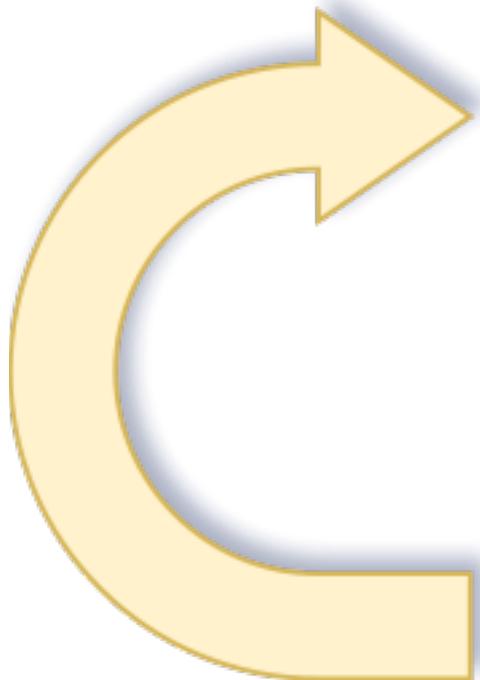


Message Consumer Deduplication



Consume Message
from Message Broker

Message Consumer Deduplication



Acknowledge Message
in Message Broker

Message Handler Deduplication

Message Handler Deduplication

```
#[Duplicated("paymentId")]
#[CommandHandler]
public function handle(MakePayment $makePayment): void
{
    // make payment
}
```

Message Handler Deduplication

```
#[Duplicated("paymentId")]
#[CommandHandler]
public function handle(MakePayment $makePayment): void
{
    // make payment
}
```

Message Handler Deduplication

```
# [Deduplicated("paymentId")]
#[CommandHandler]
public function handle(MakePayment $makePayment): void
{
    // make payment
}
```

Message Handler Deduplication

```
##[Route('/payment', 'POST')]
public function makePayment(Request $request): Response
{
    $this->commandBus->send(
        new MakePayment($request->get('amount'))
    ,
    [
        'paymentId' => $request->get('paymentId')
    ]);
}
```

Message Handler Deduplication

```
#Route('/payment', 'POST')
public function makePayment(Request $request): Response
{
    $this->commandBus->send(
        new MakePayment($request->get('amount'))
    ,
    [
        'paymentId' => $request->get('paymentId')
    ]);
}
```

Message Handler Deduplication

```
#Route('/payment', 'POST')
public function makePayment(Request $request): Response
{
    $this->commandBus->send(
        new MakePayment($request->get('amount'))
    ,
    [
        'paymentId' => $request->get('paymentId')
    ]);
}
```



Interceptors

```
final class AuthorizationInterceptor
{
    #[Before(pointcut: OrderService::class)]
    public function intercept(
        #[Header("executorId")] ?string $executorId
    ): void
    {
        if ($executorId === null) {
            throw new \InvalidArgumentException(
                message: 'Unauthorized action.'
            );
        }
    }
}
```

Interceptors

```
final class AuthorizationInterceptor
{
    #[Before(pointcut: OrderService::class)]
    public function intercept(
        #[Header("executorId")] ?string $executorId
    ): void
    {
        if ($executorId === null) {
            throw new \InvalidArgumentException(
                message: 'Unauthorized action.'
            );
        }
    }
}
```



Interceptors

```
final class AuthorizationInterceptor
{
    #[Before(pointcut: OrderService::class)]
    public function intercept(
        #[Header("executorId")] ?string $executorId
    ): void
    {
        if ($executorId === null) {
            throw new \InvalidArgumentException(
                message: 'Unauthorized action.'
            );
        }
    }
}
```



Interceptors

```
final class AuthorizationInterceptor
{
    #[Before(pointcut: OrderService::class)]
    public function intercept(
        #[Header("executorId")] ?string $executorId
    ): void
    {
        if ($executorId === null) {
            throw new \InvalidArgumentException(
                message: 'Unauthorized action.'
            );
        }
    }
}
```



Interceptors Pointcuts

Interceptors Pointcuts

```
#[Before(pointcut: CommandBus::class)]
```

Interceptors Pointcuts

```
# [Before(pointcut: CommandBus::class)]
```

```
# [Before(pointcut: CommandHandler::class)]
```

Interceptors Pointcuts

```
# [Before(pointcut: CommandBus::class)]
```

```
# [Before(pointcut: CommandHandler::class)]
```

```
# [Before(pointcut: "App\Application\*")]
```

Custom Attribute Pointcut

Custom Attribute Pointcut

```
#[\Attribute]
final class RequiredRole
{
    public function __construct(public readonly string $role)
    {
    }
}
```


Custom Attribute Pointcut



```
#[\Attribute]
final class RequiredRole
{
    public function __construct(public readonly string $role)
    {
    }
}
```


Custom Attribute Pointcut

```
# [RequiredRole("ROLE_USER")]
#[CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```


Custom Attribute Pointcut

```
# [RequiredRole("ROLE_USER")]
#[CommandHandler]
public function placeOrder(
    PlaceOrder $placeOrder,
    OrderRepository $orderRepository): void
{
    $order = Order::create($placeOrder->productName);
    $orderRepository->save($order);
}
```


Custom Attribute Pointcut

```
# [Before(pointcut: RequiredRole::class)]
public function intercept(
    #[Header("executorRole")] string $role,
    RequiredRole $requiredRole
): void
{
    if ($role !== $requiredRole->role) {
        throw new \InvalidArgumentException(
            message: 'Unauthorized action.'
        );
    }
}
```


Custom Attribute Pointcut



```
#[Before(pointcut: RequiredRole::class)]
public function intercept(
    #[Header("executorRole")] string $role,
    RequiredRole $requiredRole
): void
{
    if ($role !== $requiredRole->role) {
        throw new \InvalidArgumentException(
            message: 'Unauthorized action.'
        );
    }
}
```


Custom Attribute Pointcut

```
# [Before(pointcut: RequiredRole::class)]
public function intercept(
    #[Header("executorRole")] string $role,
    RequiredRole $requiredRole
): void
{
    if ($role !== $requiredRole->role) {
        throw new \InvalidArgumentException(
            message: 'Unauthorized action.'
        );
    }
}
```

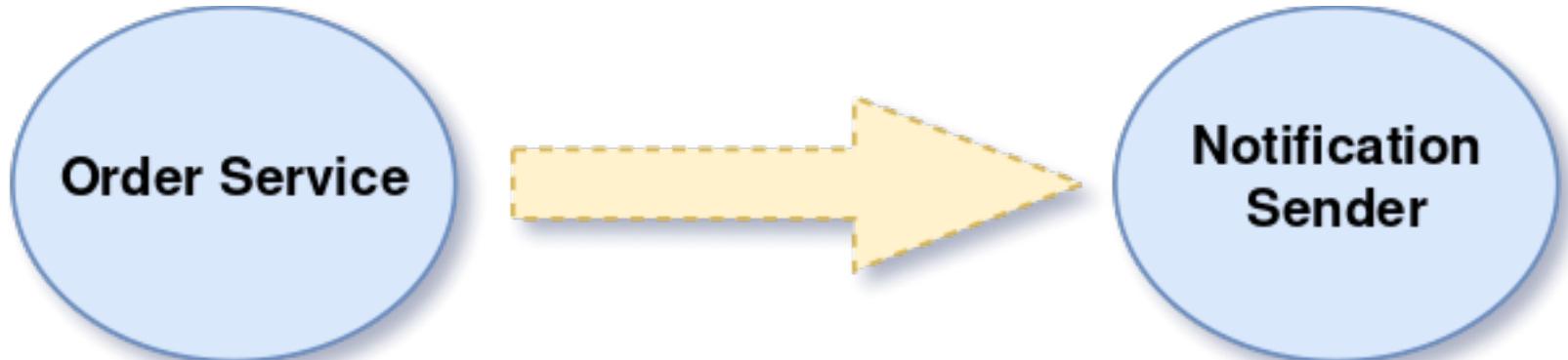


Custom Attribute Pointcut

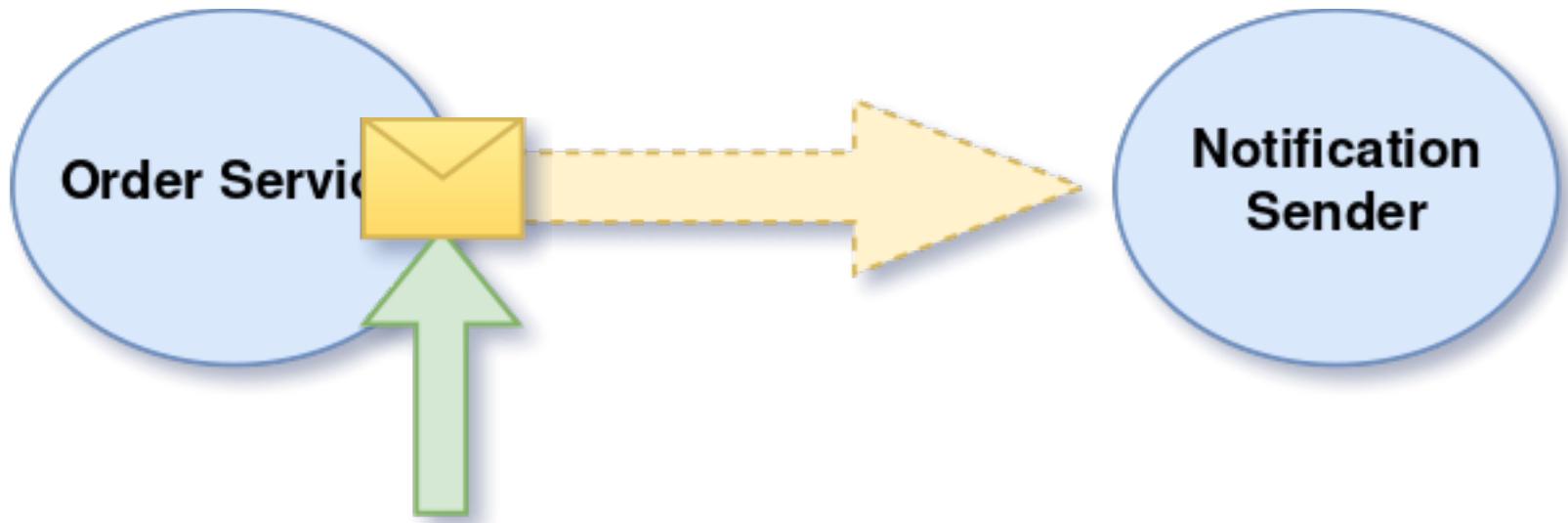
```
# [Before(pointcut: RequiredRole::class)]
public function intercept(
    #[Header("executorRole")] string $role,
    RequiredRole $requiredRole
): void
{
    if ($role !== $requiredRole->role) {
        throw new \InvalidArgumentException(
            message: 'Unauthorized action.'
        );
    }
}
```



Interceptor Types

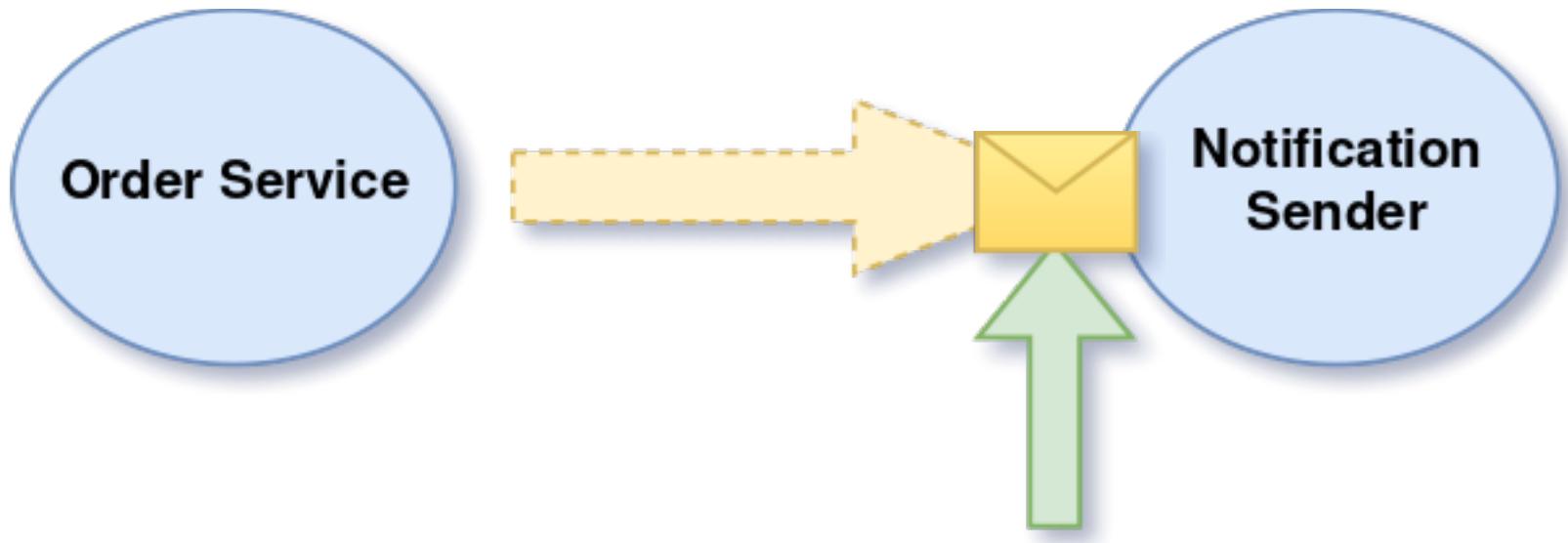


Interceptor Types



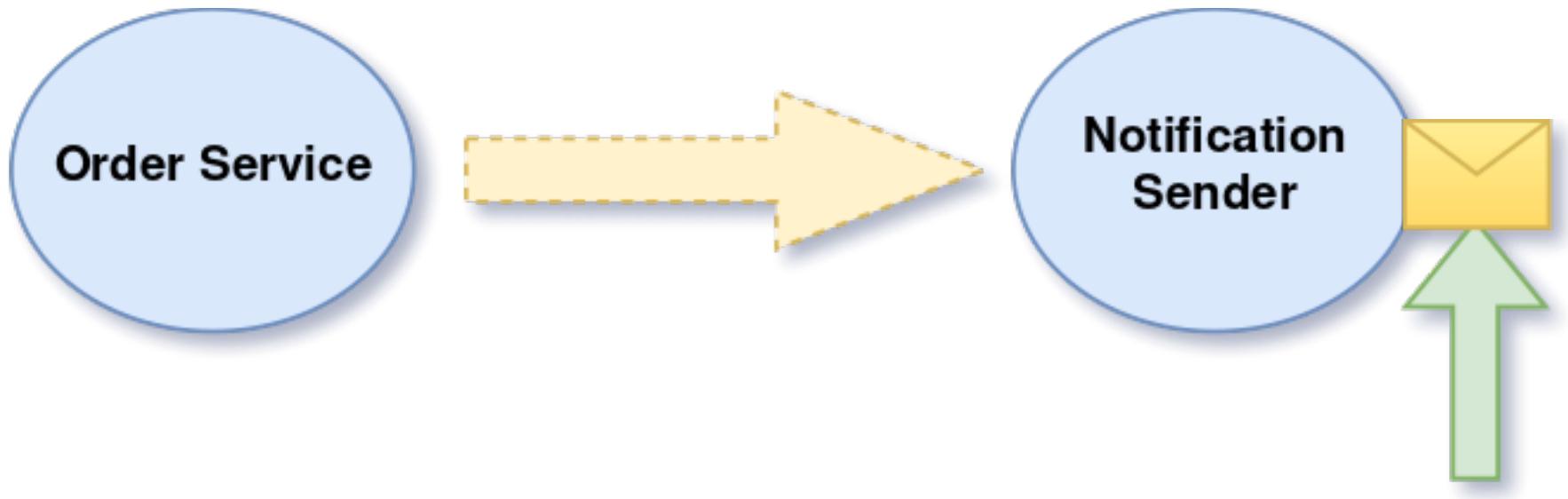
Presend

Interceptor Types



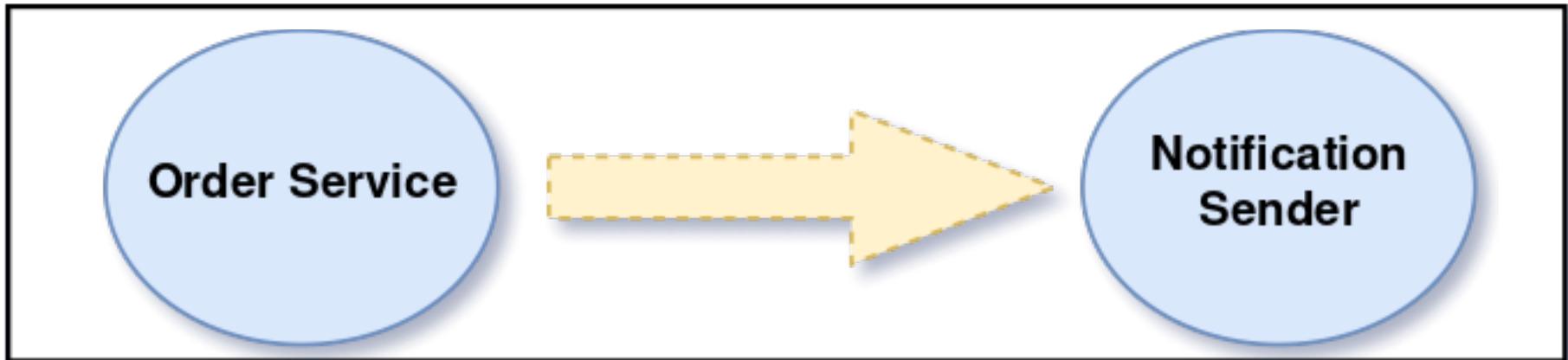
Before

Interceptor Types



✓ After

Interceptor Types



Schedulers

```
final class ReminderSender
{
    #[Scheduled(endpointId: 'reminderSender')]
    #[Poller(fixedRateInMilliseconds: 1000)]
    public function sendReminders(): void
    {
        // send reminders
    }
}
```

Schedulers

```
final class ReminderSender
{
    #[Scheduled(endpointId: 'reminderSender')]
    #[Poller(fixedRateInMilliseconds: 1000)]
    public function sendReminders(): void
    {
        // send reminders
    }
}
```

Schedulers

```
final class ReminderSender
{
    #[Scheduled(endpointId: 'reminderSender')]
    #[Poller(fixedRateInMilliseconds: 1000)]
    public function sendReminders(): void
    {
        // send reminders
    }
}
```

Inbuilt Repositories

Inbuilt Repositories



Doctrine ORM Repository

Inbuilt Repositories



Doctrine ORM Repository



Eloquent Repository

Inbuilt Repositories



Doctrine ORM Repository



Eloquent Repository



Document Store Repository

Inbuilt Repositories



Doctrine ORM Repository



Eloquent Repository



Document Store Repository



Prooph Event Sourcing
Repository

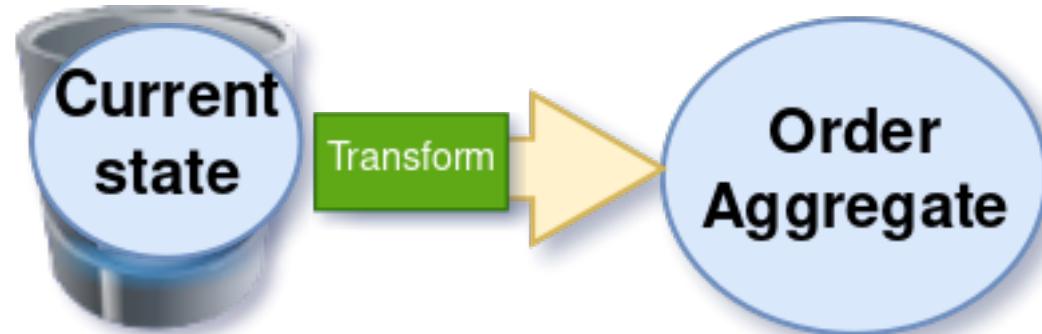
Storing Aggregates



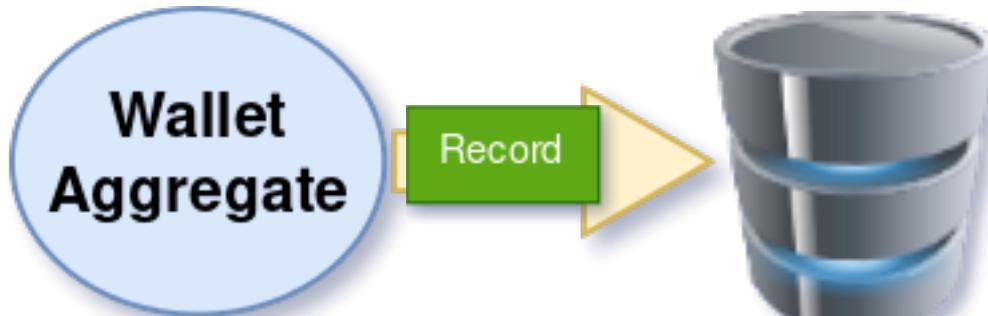
```
#[Aggregate]
final class Order
{
    #[AggregateIdentifier]
    private string $orderId;
    private bool $isCancelled = false;
    private string $reason;

    #[CommandHandler]
    public function cancel(CancelOrder $command): void
    {
        $this->isCancelled = true;
        $this->reason = $command->reason;
    }
}
```

Storing Aggregates

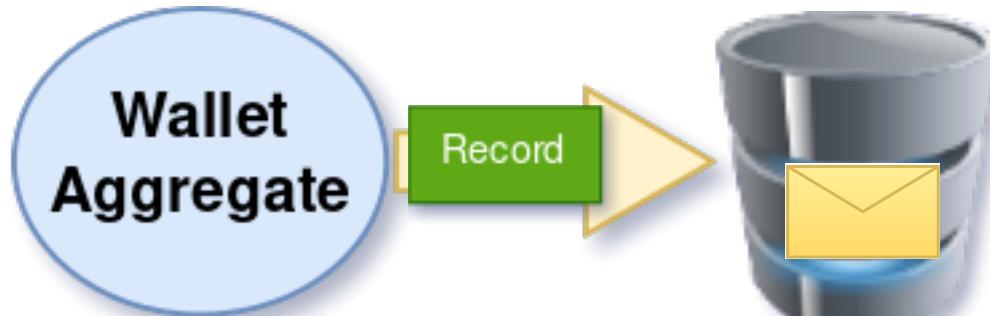


Storing Aggregates

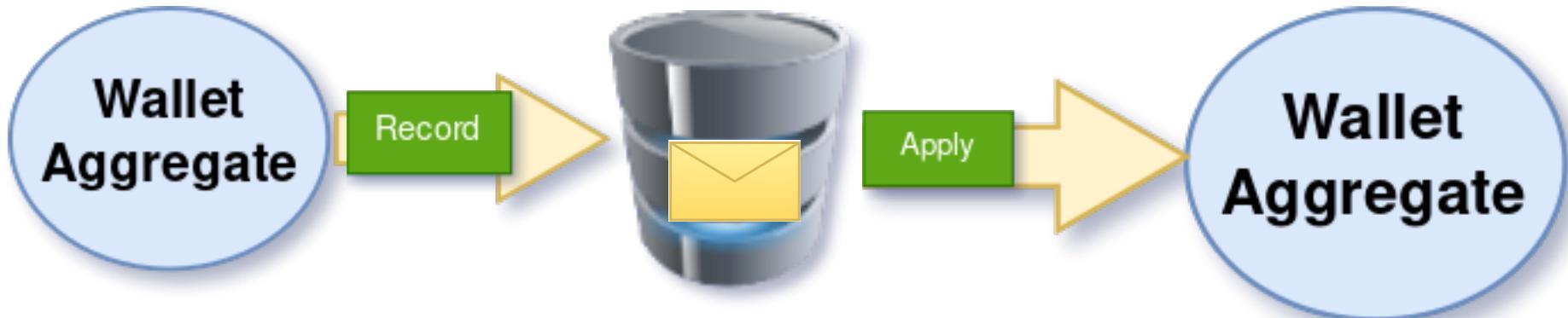


```
# [CommandHandler]
public function deposit(DepositMoney $command): array
{
    return [new MoneyWasDeposited($command->walletId, $command->amount)];
}
```

Storing Aggregates

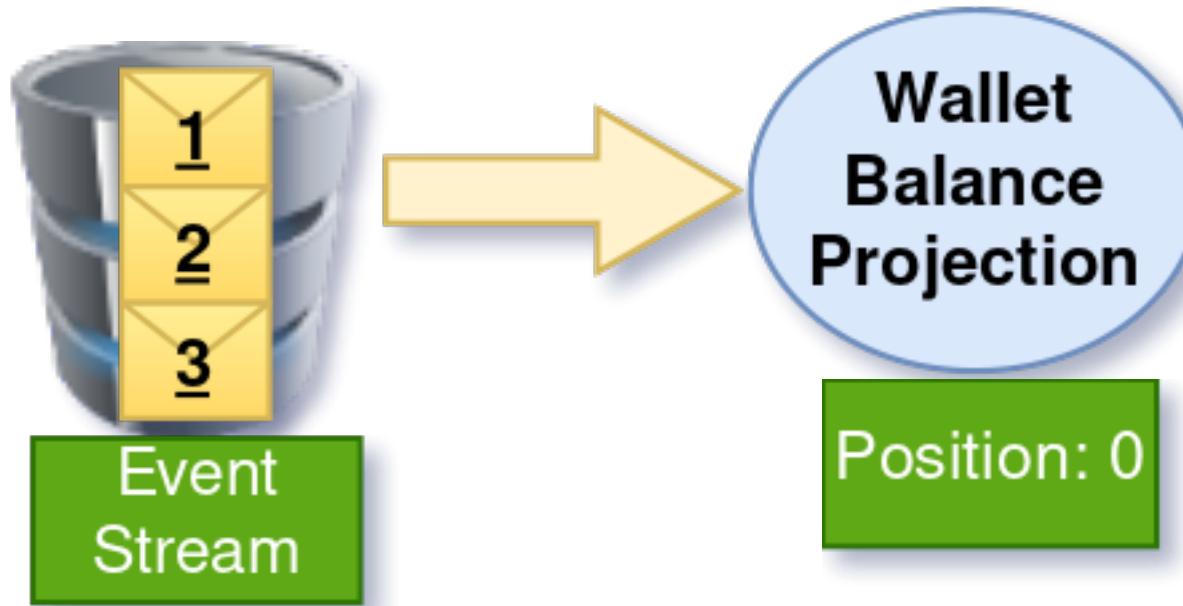


Storing Aggregates

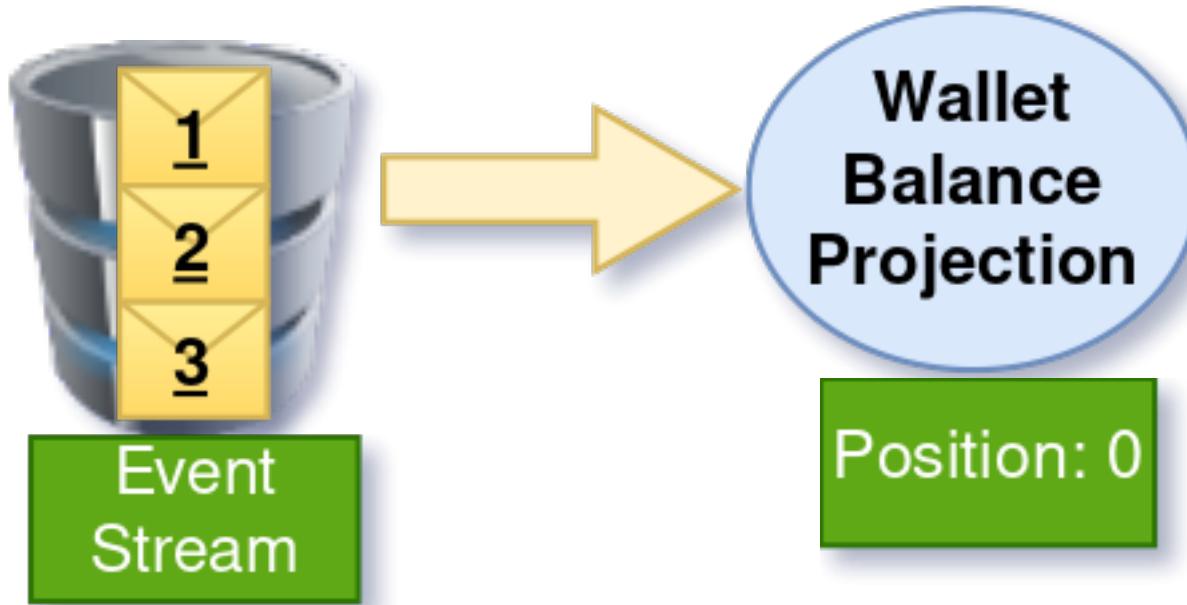


```
# [EventSourcingHandler]
public function applyMoneyWasDeposited(MoneyWasDeposited $event): void
{
    $this->balance += $event->amount;
}
```

Projections

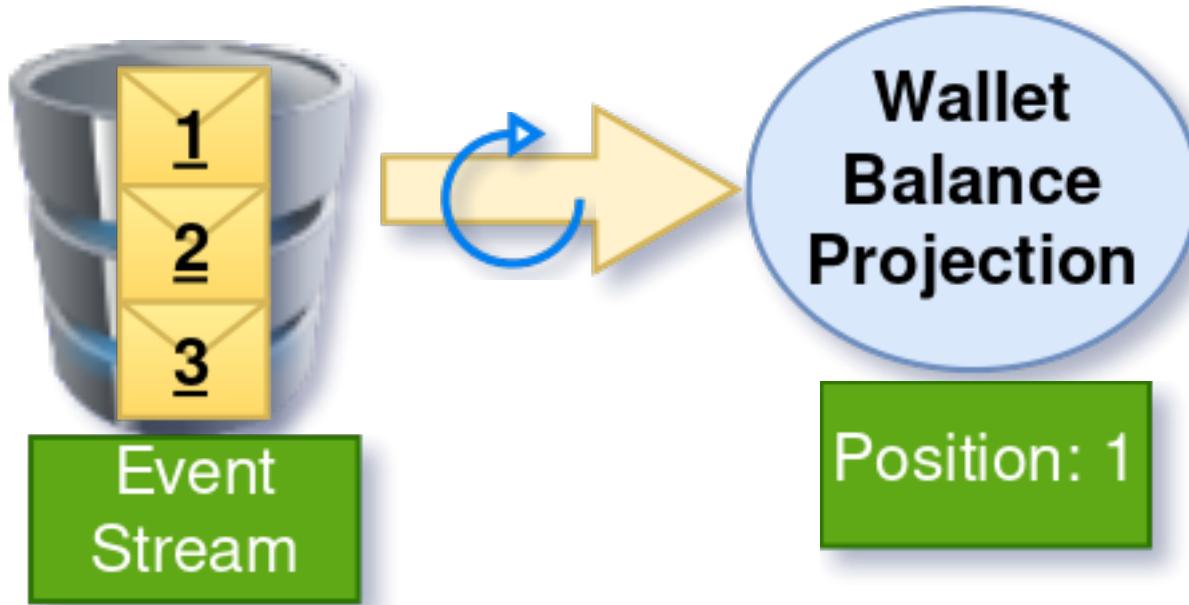


Projections



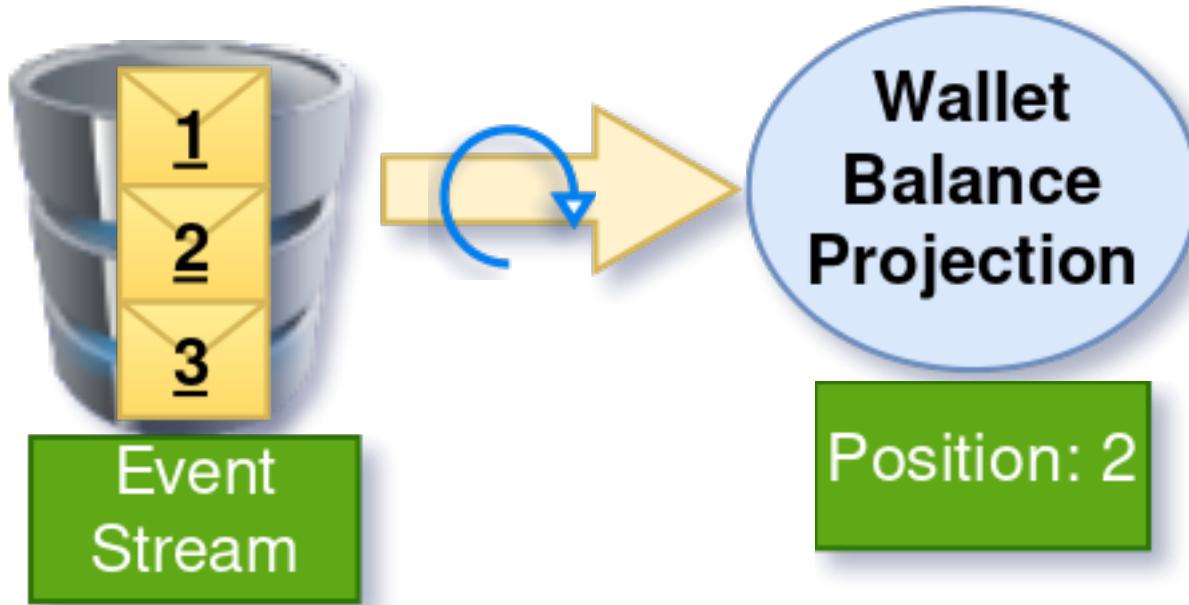
```
#[Projection("current_balance", Wallet::class)]
final class CurrentBalanceProjection
{
    #[EventHandler]
    public function whenMoneyWasDeposited(MoneyWasDeposited $event): void
    {
        $this->addToWallet($event->walletId, $event->amount);
    }
}
```

Projections



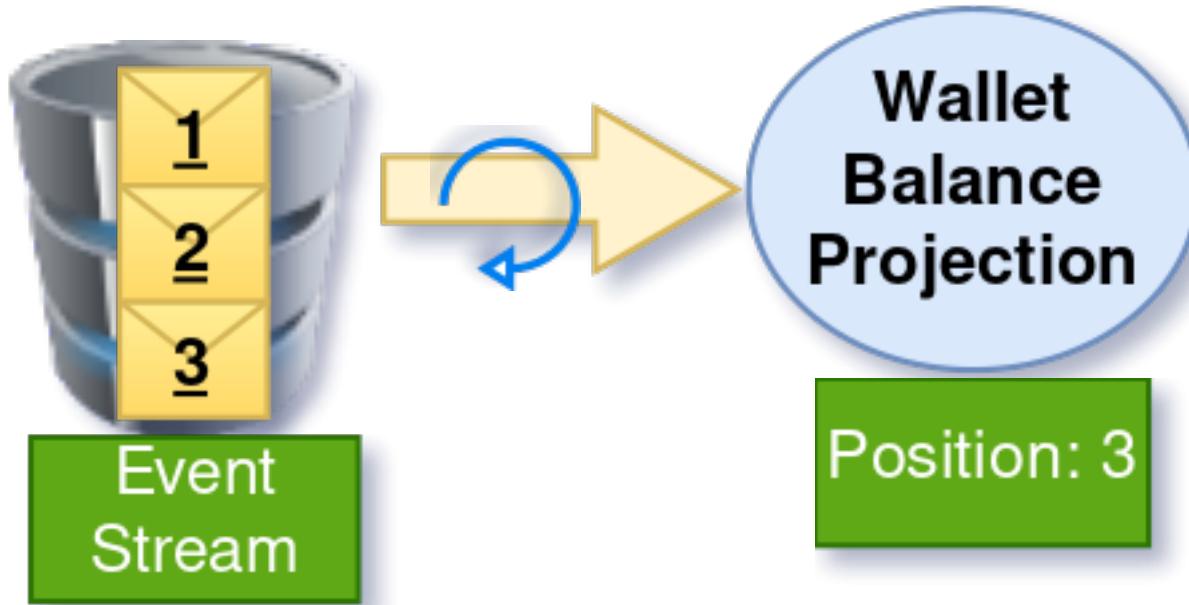
```
#[Projection("current_balance", Wallet::class)]
final class CurrentBalanceProjection
{
    #[EventHandler]
    public function whenMoneyWasDeposited(MoneyWasDeposited $event): void
    {
        $this->addToWallet($event->walletId, $event->amount);
    }
}
```

Projections



```
#[Projection("current_balance", Wallet::class)]
final class CurrentBalanceProjection
{
    #[EventHandler]
    public function whenMoneyWasDeposited(MoneyWasDeposited $event): void
    {
        $this->addToWallet($event->walletId, $event->amount);
    }
}
```

Projections



```
#[Projection("current_balance", Wallet::class)]
final class CurrentBalanceProjection
{
    #[EventHandler]
    public function whenMoneyWasDeposited(MoneyWasDeposited $event): void
    {
        $this->addToWallet($event->walletId, $event->amount);
    }
}
```



Questions?